VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zmenšení databáze s ohledem na vytížení

Database Subsetting with Respect to Workload

2018

Denis Mělišek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student:	**Bc. Denis Mešišek**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	2612T025 Informatika a výpočetní technika
Téma:	Zmenšení databáze s ohledem na vytížení Database Subsetting with Respect to Workload
Jazyk vypracování:	čeština

Zásady pro vypracování:

Zmenšení databáze je typický databázový problém, který se využívá pro vygenerování smysluplné testovací kolekce z ostrých dat. Existuje několik nástrojů, které ji umožňují provést a zachovat referenční integritu dat. Vstupem do těchto nástrojů je obvykle procento dat, které chceme mít na výstupu. Cílem této práce je vytvořit nástroj, který umožní zmenšení databáze s ohledem na vytížení definované operacemi nad databází. Hlavním kritériem je, aby dotazy ve vytížení nad zmenšenou databází vracely stejné výsledky jako nad databází původní. Tato práce bude navazovat na prototyp vytvořený v rámci semestrálního projektu a hlavní rozšíření realizované v rámci diplomové práce budou:
1. Podpora poddotazů (ve všech částech SELECT).
2. Podpora dotazů obsahující konstrukce IN, EXISTS, ALL a ANY.
3. Podpora dotazů obsahující konstrukce GROUP BY a HAVING.
4. Provedení zmenšení databáze tak, aby zmenšená databáze neobsahovala žádná data navíc, tzn. žádná data po jejichž odstranění by vytížení vracelo stejné výsledky.

Práce bude probíhat v následujících krocích:
1. Analýza a návrh algoritmů, které budou řešit situace pro požadované konstrukce.
2. Implementace algoritmů.
3. Důkladné testování fungování celého řešení na netriviální množině dotazů.
4. Porovnání s podobnými nástroji.

Seznam doporučené odborné literatury:

[1] Dokumentace k DataBee, URL: https://www.databee.com/dtb_manual.htm

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018

.....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2018

.....

Rád bych poděkoval panu Ing. Radimovi Bačovi, Ph.D. za vedení mé práce a za velkou snahu, podporu, trpělivost a laskavost při vypracovávání diplomové práce.

Abstrakt

Diplomová práce je zaměřena na zmenšení relační databáze. Teoretická část práce je rozdělena na dvě hlavní části. V první části se zaměřuje na syntaxi příkazu SQL Select, který se běžně používá například pro získávání dat z databáze. Kapitola popisuje části SQL Select, jeho strukturu, technologie a vysvětluje, jakou podstatu hraje v naší problematice. V druhé části seznamuje s problematikou zmenšení databáze. Vysvětluje tuto problematiku a popisuje nástroje, kterými ji lze řešit. V praktické části popisuje naimplementovanou aplikaci, která řeší tento problém. Vysvětluje jeho strukturu, funkcionalitu a samotný algoritmus zmenšení databáze. Závěrečná část se věnuje testování aplikace a následné analýze naměřených výsledků.

Klíčová slova: relační databáze, SQL dotaz, zmenšení databáze

Abstract

Master thesis is focused on a relational database subsetting. Theoretical part of this thesis is divided into two main parts. In the first part thesis focuses on SQL Select syntax, which can be used for selecting or getting data from a database. This chapter describes SQL Select parts, its structure and technology and explains what role it plays in theme of this thesis. In the second part thesis familiarizes us with database subsetting and explains it and describes tools, which can solve this kind of problem. In practical part thesis describes application, which was created for dealing with this kind of problem. It explains structure of the application and its functionality and tries to clarify an algorithm, which solves the database subsetting. Final part of the thesis devotes to testing the application and analyzing measured results.

Key Words: relational database, SQL select, database subsetting

Obsah

Seznam použitých zkratek a symbolů

SQL	–	Structured Query Language
SEQUEL	–	Structured English Query Language
ANSI	–	American National Standards Institute
SQUARE	–	Specifying Queries As Relational Expressions
GUI	–	Graphical User Interface

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době se využívají různá data a ty potřebujeme někam ukládat, jednou možností jsou relační databáze. Ty nejen, že skladují data, ale poskytují velikou škálu funkcí, které můžeme využít, abychom si zjednodušili práci s našimi daty.

Data jsou v relačních databázích uložené v tabulkách, které mohou být mezi sebou propojené vazbami, pomocí kterých snadněji dohledáme souvislosti mezi daty samotnými. Kromě tabulek se v databázi vyskytují i jiné struktury jako například: indexy, sekvence spouštěče a další, všechny tyto objekty můžeme nejen vytvářet, ale i upravovat či mazat. Ale nejpodstatnější věcí, kterou nám databáze nabízí, je dotazování nad samotnými daty. K tomu nejčastěji využíváme dotazovací jazyk SQL, který je uzpůsobený k vytváření SQL příkazů (Select, Insert, ...).

S plynoucím časem přibývá stále více dat, které nechceme mazat, ale naopak je všechny potřebujeme ukládat, abychom je mohli později využívat. S tímto trendem, který už je v dnešní době tak samozřejmý, přichází nové problémy, ale i možnosti, jak s těmito daty pracovat či upravovat.

To nás vede k zamýšlení, co všechno můžeme s těmito daty dělat. Možností je nespočet, můžeme je analyzovat, zneužívat, prodávat, ale i zmenšovat. S velkými daty taktéž přichází velké nároky na paměťovou kapacitu, časově náročné zpracování či obyčejný přesun z jednoho místa na druhé.

Když tento problém převedeme konkrétnější případ, tak se dostaneme k databázím. Ty se neustále plní novými daty, které ne vždy potřebujeme, ale i přesto je ukládáme, protože nikdy nevíme, zda se nebudou v budoucnosti hodit.

Nad daty provádíme neustále mnoho operací do té doby, než tyto operace netrvají příliš dlouhou dobu a my si začneme říkat, zda by se to nedalo nějakým způsobem řešit. Můžeme přijít do situace, kdy zbytečně dotazy, které pracují s velmi velkými daty a kvůli tomu, pokud není dotaz správně napsán, může trvat i vteřiny či minuty navíc. Jednou možností je zmenšit databází jen na množinu potřebných záznamů a ty si uložit samostatně, pak při stejném dotazovaní se provedou dotazy rychleji a výsledky budou stejné.

Provedení zmenšení databáze a jejich dat není jen teoretická záležitost, ale dá se provést reálně, bude mít sice několik omezení, ale v konečném výsledku budou data skutečně menší.

A proč vůbec provádět zmenšení databáze? Je několik situací, kdy se nám tohle zmenšení může hodit. Máme například obrovskou databázi, která je navíc neustále používaná a vyskytují se v ní ostrá data, která nechceme nijak poškodit a my potřebujeme část této databáze k testování aplikace, která se zrovna vyvíjí, takže není nic lehčího než si definovat zmenšenou databázi například nějakými SQL dotazy, a pak ji

„zmenšit“, čímž právě dostaneme už menší databázi s požadovanými daty a ta se pak může použít jako testovací databáze pro danou vyvíjenou aplikaci.

2 Základy SQL

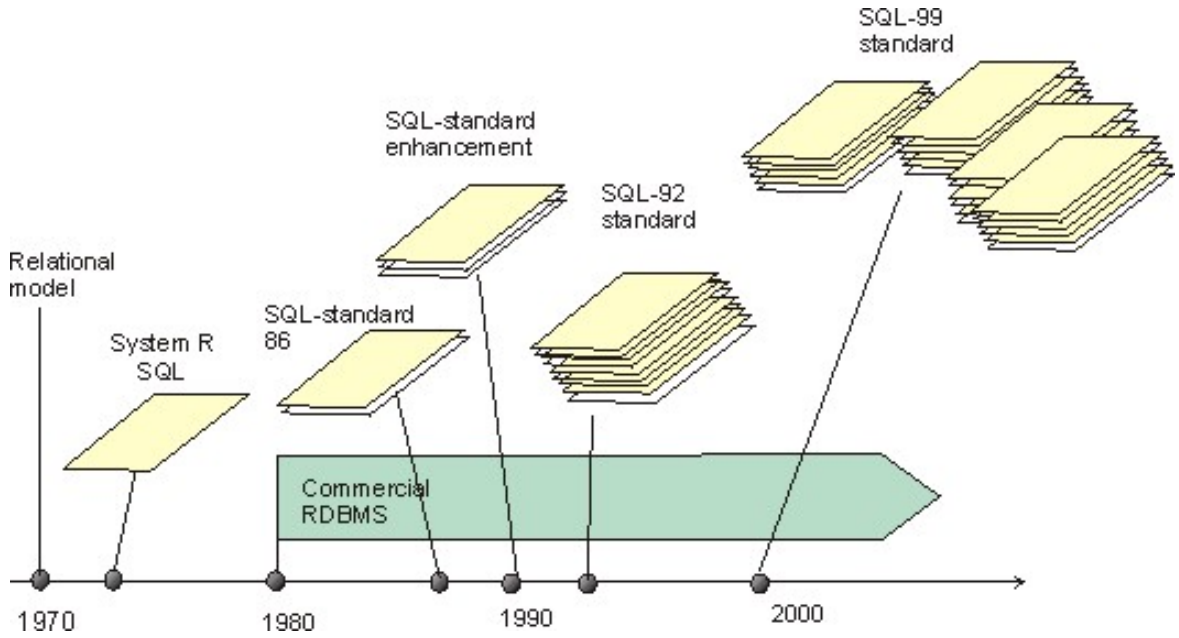
2.1 Historie

Jazyk SQL byl původně vyvinut firmou IBM v 70. letech Donaldem D. Chamberlinem a Raymondem F. Boycem po té, když se dozvěděli a nastudovali relační model z článku "A Relational Model of Data for Large Shared Data Banks", který publikoval v roce 1970 Dr. E.F. Codd.

Chamberlin and Boyce se nejdříve pokoušeli vytvořit jazyk pro relační databázi nazýván SQUARE. Byl složitý na používání kvůli své notaci, to pak vedlo k tomu, že v roce 1973 začali pracovat na novějším jazyce SEQUEL. Zkratka tohoto jazyka byla pak časem změněna na SQL.

Na konci 70. let společnost (v dnešní době nazývaná Oracle) viděla potenciál těchto konceptů a vyvinula svou vlastní databázový systém postavený na SQL za účelem jeho poskytování či prodávání jiným organizacím. V 1979 pak přišla s první komerční verzí systému využívající SQL.

V roce 1986 americký národní standardizační institut (ANSI) standardizoval SQL. Tento standart byl upraven v 1989, 1992 (SQL2), 1999 (SQL3), 2003 (SQL 2003), 2006 (SQL 2006), 2008 (SQL 2008). Standart SQL je nkědý nazýván ANSI SQL. Všechny významné relační databáze podpurují tento standart, proto můžeme SQL používat v Oracle databázích, Microsoft SQL Serveru nebo MySql databázích. [1][2]



Obrázek 1: Historie SQL [3]

2.2 Co to je SQL?

SQL je zkratka pro „Structured Query Language“, což v překladu znamená: Strukturovaný dotazovací jazyk. Je používán ke komunikaci s databází. Podle ANSI (Americký národní standardizační institut) je to standardizovaný jazyk pro správu relačních databázových systémů.

Jazyk SQL je rovněž deklarativní jazyk, který neobsahuje cykly, podmínky, procedury a další příkazy. A co si můžeme představit pod pojmem deklarativní jazyk? Ve zkratce se jedná o to, že napíšete například SQL dotaz, ve kterém zadáte, jaké záznamy chcete vrátit (zobrazit), ale už nezmiňujete, jak se to má provést.

SQL příkazy jsou používány k provádění různých úkolu jako například: úprava, vkládání, mazání dat v databázi nebo získávání dat z databáze. Můžeme si uvést několik databázových systémů: Oracle, Sybase, Microsoft SQL Server, Access, Ingres a další.

Ačkoliv většina databázových systémů používá SQL, tak značná část z nich má také své vlastní nadstavby, které jsou typické pro jejich systém. Avšak standartní SQL příkazy jakožto: Select, Insert, Update, Delete nebo Create mohou být využívaný ve všech systémech k provedení téměř všeho, co potřebujeme dělat v databázi. [4]

2.3 Základní rozdělení SQL

1. DDL (jazyk pro definici dat)
    - vytváření databázových objektů (CREATE)
    - změna definice databázových objektů (ALTER)
    - mazání databázových objektů (DROP)
2. DML (jazyk pro manipulaci s daty)
    - výběr záznamů (SELECT)
    - vkládání záznamů (INSERT)
    - mazání záznamů (DELETE)
    - úprava záznamů (UPDATE)
3. DCL (jazyk pro správu dat)
    - poskytnutí práva (GRANT)
    - odebrání práva (REVOKE)
4. TCC (příkazy pro řízení transakcí)
    - uložení změn v databázi (COMMIT)

- rušení změn provedené transakcí (ROLLBACK)
- bod návratu (SAVEPOINT)

[5]

3 SELECT

3.1 Syntaxe

```
SELECT [DISTINCT]
{ { agregační funkce | výraz [AS název sloupce] }.,..}
| {kvalifikátor.*} | *
INTO specifikace cíle.,..
FROM { {název tabulky [AS] [alias]
[ (název sloupce.,..) ] }
| {poddotaz |spojová tabulka | konstruktor hodnot tabulky
| { název tabulky } } [AS] alias [ (název sloupce.,..) ] } } .,..
[ WHERE predikát]
[ GROUP BY { název tabulky | alias }.název sloupce]
[ COLLATE název řazení ] ]
[ HAVING predikát ]
[ { UNION | INTERSECT | EXCEPT }
[ALL] [CORRESPONDING] [ BY (název sloupce.,..) ] ]
příkaz select | { název tabulky }
| konstruktor hodnot tabulky ]
[ ORDER BY { { výstupní sloupec [ ASC | DESC ] }.,..}
| { { kladné cele číslo [ ASC | DESC ] }.,..}; [6]
```

3.2 Použití

Příkazem SELECT se vytvářejí dotazy, které ve většině případů požadují nějaké informace z databáze. Uživatel, který vytváří tyhle dotazy, musí mít určitá práva a to především právo SELECT ke všem tabulkám, ze kterých bude požadovat data.

Dotazy můžeme použít samostatně na výběr dat z jedné či více tabulek. Taktéž mohou tvořit část definic kurzorů nebo pohledů. Další velké použití dotazů jsou poddotazy, kterými většinou vytváříme hodnoty, které se například použijí uvnitř vnějších příkazů.

Výsledek dotazu je v podstatě tabulka a klauzule SELECT nastavuje výstupní sloupce této tabulky. Výstupní sloupce můžou pocházet přímo z tabulek zadaných v dotazu, nebo jsou odvozené pomocí hodnot z těchto tabulek anebo se jedna o výrazy, které nevyužívají obsah tabulky.

Následující seznam klauzulí vyskytujících se v SELECT dotazu zobrazuje, v jakém pořadí jsou ve skutečnosti vyhodnocovány:

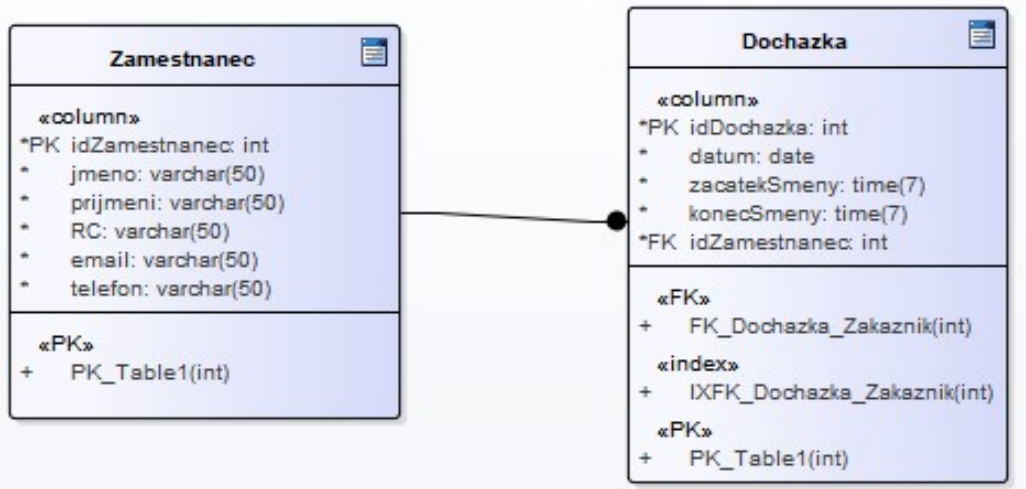
1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. UNION nebo EXCEPT
7. INTERSECT
8. ORDER BY
9. INTO

[6]

Na příkladu si objasníme a stručně popíšeme, jak se celkem jednoduchý dotaz z hlediska jeho části vykonává a na co tedy myslet při psaní dotazu.

Příklad 1

Mějme databázi obsahující pouze 2 tabulky, které budou mít mezi se vazbu. Tohle nám pro naše pochopení bude bohatě stačit.



Obrázek 2: SQL dotaz

Dále si vytvoříme dotaz, který nám zobrazí zaměstnance, kteří mají email na seznamu.cz a navíc byli v práci minimálně 15 dní v lednu. Tento dotaz by šel napsat několika způsoby, ale my vybereme takový, na kterém se dá lehce vysvětlit, v jakém pořadí se dotaz vykonává. Následný dotaz je spustitelný v databázích na MS SQL serverech.

```
SELECT z.jmeno, z.prijmeni, COUNT(*) "Pocet dni v praci"
FROM zamestnanec z JOIN dochazka d on z.idZamestnanec = d.idZamestanec
   and DATEPART(MONTH, d.datum) = 1
WHERE z.email like '%@seznam.cz'
GROUP BY z.jmeno, z.prijmeni
HAVING COUNT(*) >= 15
ORDER BY z.prijmeni
```

Výpis 1: Testovací dotaz

Nejdříve se musíme zamyslet, odkud budeme brát záznamy. Ty najdeme ve dvou tabulkách, které jsme v našem případě spojily pomocí slova „JOIN“, více o spojení si povíme později. Dále si všimněme, že ke spojení se použijí podmínky, ty nám říkají, dle čeho dojde ke spojení záznamů, a tedy klauzule FROM se provádí jako první v dotazu a jejím výsledkem je jakási prozatímní množina dat ať už z jedné nebo více tabulek. Pak přichází na řadu Klauzule WHERE, ve které se nacházejí podmínky, které nám filtrují množinu dat, v příkladu vidíme, že se vyfiltruji záznamy, jejichž sloupec email odpovídá dané podmínce. Následuje klauzule GROUP BY, která zjednodušeně řečeno seskupuje záznamy do skupin dle zadaných sloupců. Pak může přijít na řadu klauzule HAVING, která dané skupiny filtruje dle jejich výsledku, v příkladu je výsledek počet dni, které jsme získali pomocí agregační funkce COUNT. Až teď se dostaneme ke klauzuli SELECT a v ní vybereme výstupní sloupce, které chceme zobrazit ve výsledku a jako poslední se provede seřízení záznamů dle zadaných sloupců a nakonec dostaneme výslednou množinu záznamů.

Musíme se tedy nejdříve rozmyslet, jaké všechny tabulky budeme potřebovat, zda je nutné je spojovat a jak, po případě nepoužít raději poddotaz. Pak jestliže spojujeme data, zda by nebylo vhodné použít filtry během spojování a ne až v klauzuli WEHRE. Důležité si i uvědomit, že pokud budeme seskupovat data, tak si uvědomit, že některé podmínky můžeme použít jen v klauzuli HAVING (agregační funkce) a nikoliv v klauzuli WHERE, proto bychom nemohli například „COUNT(*) >= 15“ použít v klauzuli WHERE, protože bychom dostali chybu.

V následující kapitole si popíšeme podrobněji všechny části dotazu, abychom si mohli udělat představu, jak a z čeho všeho se může dotaz skládat.

3.3 Klauzule SELECT

Klauzule SELECT uvádíme jako první, ale prvním logickým krokem při zpracování dotazu to není. Máme-li všechny zdrojové řádky příkazu, ze kterých se bude odvozovat výstup, tak klauzule SELECT jen stanovuje, jaké sloupce z dané množiny výsledků budou výstupní, a tedy budou zobrazeny ve výsledku. Na výstupu se můžou vyskytovat přímo sloupce nebo se sloupce mohou objevovat v agregačních funkcích či výrazech, které mohou nabývat různých datových typů jako například: řetězec, celé nebo desetinné číslo, datum, čas a jiné.

Objevuje-li se klíčové slovo DISTINCT na začátku dotazu (hned za klíčovým slovem SELECT), tak to znamená, že ve výsledné množině výstupních dat se najdou duplicitní záznamy a ponechají se z nich jen jeden, tahle situace výskytu duplicit může nastat například při spojení tabulek, kde dostaneme duplicitní hodnoty kvůli tomu, že ve spojené tabulce je více záznamů pro každý záznam v původní tabulce. Takových případů, kdy narážíme na výskyt duplicit je mnoho, ale většinou je to špatně vytvořeným dotazem a samozřejmě, že je jednodušší použít slovo DISTINCT, ale pokud nechceme být jako všichni ostatní a budeme se řídit některými zásady při vytváření dotazu, tak bychom vždy našli způsob, jak napsat dotaz tak, abychom nemuseli se na konci zbavovat duplicitních řádku.

V klauzuli SELECT může dále obsahovat:

- Znak hvězdička (*), což znamená, že ve výsledku budou zobrazeny všechny sloupce ze všech tabulek v takovém pořadí, v jakém jsou uvedené v tabulkách uvedené. Použití hvězdičky není někdy špatné, ale ve většině případů se doporučuje vypisovat sloupce ručně.

- Kvalifikátor s hvězdičkou (kvalifikátor.*), který vyprodukuje posloupnost všech sloupců z tabulky, ke které kvalifikátor je přiřazený. Kvalifikátor se v dotazu shoduje s názvem alias tabulky a dle toho se pozná, ke komu patří, to vede i dalšímu použití, kde použijeme kvalifikátor jako „prefix“ k názvu sloupce. Například u příkladu 1 jsme si mohli povšimnout jedno ze sloupců (z.nazev), kde vidíme „z“ jakožto kvalifikátor a víme, že patří tabulce s názvem „zamestnanec“, protože má stejnojmenný alias „z“.

- Agregační funkce, Jedná se o funkci, která vrací pouze jednu hodnotu a ta se odvozuje ze skupiny sloupcových hodnot - například COUNT, MIN, nebo SUM (počet, součet nebo minimum).

- Výraz, který obsahuje více atomických hodnot, které jsou nějakým způsobem propojené, a jejím výsledkem je jedna výstupní hodnota. Například si můžeme uvést, že neuvedeme jen sloupec cena, ale cena * 1,21, což nám dá výraz, který vynásobí hodnotu ve sloupci cena.

- Vnořený dotaz, jehož jedním z hlavních požadavků je, že musí vracet právě jeden výsledek.
- AS, což je pro indikaci aliasu, ale není povinný se tam psát. Může se vyskytovat jak za názvem sloupce, vnořeného dotazu, tak i za výrazem.
- TOP n, kde „n“ je celé číslo představující maximální počet řádků, které chceme zobrazit na výstupu. Toto omezení nejde použít například s agregačními funkcemi. [6]

3.4 Klauzule INTO

Tato klauzule je typická pro SQL Server. Její hlavní úkol spočívá v tom, že kopíruje data do nové tabulky. Nová tabulka obsahuje sloupce, které nastavíme do dotazu, ve kterém se nachází klauzule INTO.

```
SELECT * INTO zamestnanec_zaloha
FROM zamestnanec
```

Výpis 2: Dotaz jehož výsledek se uloží do nové tabulky

Na této ukázce vidíme, že tuto klauzuli můžeme například použít pro zálohování specifických dat. Od klasické zálohy dat v tabulce se to liší v tom, že tady si můžeme sestavit dotaz, který nám vrátí specifické záznamy i z více tabulek a uloží do jedné tabulky.

3.5 Klauzule FROM

V této části dotazu nalezneme všechny zdrojové tabulky, které budeme v daném dotazu využívat. Samozřejmě se nejedna jenom o holý výpis tabulek ale jako tabulky mohou být též považovány:

- Pohledy, jejichž názvy se mohou podobat názvům tabulek a taky se k nim přímo přistupuje. V podstatě se jedna o uložené dotazy vracející určité záznamy.
- Vnořené dotazy, nebo jinak řečeno, tabulky odvozené nějakým poddotazem.
- Spojení.

Za všechny výše uvedené možnosti, jak zapsat tabulku, můžete uvést aliasy, čímž docílíte toho, že budete moci přistupovat k určité tabulce, tohle se nejvíce využívá v situaci, kdy máme v dotazu více tabulek, a pak můžeme narazit na problém stejně nazvaných sloupců, což by při provedení dotazu mohlo být nejednoznačné, a tudíž bychom dostali chybu, proto při přístupu ke sloupcům za pomocí aliasu se nám to stát nemůže, protože v jedna tabulce musí být názvy sloupců unikátní. [6]

3.6 Spojení (JOIN)

Nacházejí-li se v klauzuli FROM názvy dvou a více tabulek, jsou všechny implicitně spojené. To znamená, že se odvozují všechny možné kombinace řádků. Toto zřetězení řádků (z každé tabulky) nám vytvoří tabulku, na kterou budou aplikovány všechny další operace v dotazu. Takové zřetězené tabulce se také říká kartézský součin nebo křížové spojení.

Častokrát chceme většinu řádků odstranit a ponechat si jen některá data, která mají mezi sebou nějakou vazbu. Může se jednat typicky o nějaký vztah vyjádřený nějakou relací. Dá se to vyřešit pomocí klauzule WHERE. Alternativní způsob je založen na zabudovaných operátorech spojení, při kterém se celkové spojení provede v klauzuli FROM. Výsledek se po té považuje za odvozenou tabulku určenou pro další zpracování.

Obě techniky se dá kombinovat, ale udělá to více zmatku než užitku, proto se doporučuje použít vždy jen jeden z nich.

Podívejme se na pár zabudovaných operátoru spojení, které bychom mohli použít v klauzuli FROM:

- křížové spojení

```
tabulka A CROSS JOIN tabulka B
```

- přirozené spojení

```
tabulka A NATURAL [typ spojení] JOIN tabulka B
```

- sjednocené spojení

```
tabulka A UNION JOIN tabulka B
```

- specifikované spojení

```
tabulka A [typ spojení] JOIN tabulka B
{ ON predikát } | { USING (název sloupce, ..) }
```

Dále můžeme použít argumenty typu spojení (INNER, LEFT, . . .), ty blíže kvalifikují spojení, která používají přirozené nebo specifické spojení. Tato spojení ve smyslu teorie i standartu spadají do následujících kategorií, kterým se, jak už dříve bylo zmíněno, říká typy spojení. Ve všech můžeme použít klíčové slovo OUTER, které je volitelné a nemá ve spojení s typem spojení žádný účinek, pouze pokud je samostatně:

- **INNER** (vnitřní), tenhle typ je defaultní, pokud napíšeme ve spojení pouze JOIN a znamená, že se do výsledku dostanou jen řádky, které mají svůj odpovídající protějšek ve druhé tabulce.

- **LEFT [OUTER]** (levé), které znamená, se do výsledku dostanou všechny řádky z tabulky A a k nim se přidají odpovídající řádky z tabulky B, přičemž k řádkům, ke kterým se nenašli odpovídající protějšky, se doplní hodnoty NULL. Tohle si pak ukážeme v tabulce 2
- **RIGHT [OUTER]** (pravé) je opakem předchozího spojení (LEFT). Jediný rozdíl je v tom, že se do výstupu dostanou všechny řádky z tabulky B a jestliže se nenašel odpovídající řádek v tabulce A, tak se doplní hodnotami NULL. Proto říci, že:

```
tabulka A LEFT JOIN tabulka B = tabulka B RIGHT JOIN tabulka A
```

- **FULL [OUTER]** je kombinace levého a pravého spojení. Do výstupu dostaneme všechny řádky z tabulek A i B. Tam, kde se odpovídající protějšek nenajde, tak se doplní NULL hodnotami.
- Specifikované spojení používající **ON** nebo **USING**. Obě tyto možnosti mohou využívat všechny předešle typy spojení:
  - **USING** je v podstatě přirozené spojení, kde uvedete sloupce, které se mají ve spojení použít. Podmínkou je, že všechny použité sloupce se musejí vyskytovat v obou tabulkách, musí mít tedy stejný název. Tomuto druhu spojení se říká pojmenované sloupcové spojení. Tenhle typ spojení nepodporuje MS SQL Server.
  - **ON** dovoluje specifikovat nějaký predikát (podmínku). Kombinace řádků, které této podmínce vyhovují, se vyberou do výsledku. Standart tomu říká spojovací podmínka, i když se všechna spojení se provádí pomocí predikátů.

Všechny tyto spojení si ukážeme na příkladech, ve kterých si přibližně ukážeme i popíšeme, jaké rozdíly očekávat při používaní různých spojení, budeme používat stejnou databázi o dvou tabulkách (viz obr 2). V tabulce zaměstnanec máme 4 záznamy a v tabulce docházka 7 záznamů.

Křižové spojení

Při použití úplného křížového spojení dostaneme celkem 28 řádku, protože se provede spojení každého řádku s každým (čtyři záznamy se vynásobí sedmi). V následujícím výstupní tabulce nebudu zobrazovat všechny záznamy, ale jen část, pro představu to bude stačit.

```
zamestnanec z CROSS JOIN dochazka d
```

Tabulka 1: Část křížového spojení

Z.id	Z.jmeno	Z.prijmeni	D.cislo	D.datum
1	Tomáš	Starý	1	22.01.2018
1	Tomáš	Starý	2	25.02.2018
1	Tomáš	Starý	3	13.01.2018
1	Tomáš	Starý	4	25.02.2018
1	Tomáš	Starý	5	25.01.2018
1	Tomáš	Starý	6	15.03.2018
1	Tomáš	Starý	7	22.04.2018
2	Josef	Barevný	1	22.01.2018
2	Josef	Barevný	2	25.02.2018

Levé vnější spojení

U levého spojení si můžeme všimnout dříve zmiňovaných NULL hodnot, které se doplní v řádku, který nemá odpovídající protějšek.

```
zamestnanec z LEFT OUTER JOIN dochazka d
```

Tabulka 2: Levé vnější spojení

Z.id	Z.jmeno	Z.prijmeni	D.cislo	D.datum
1	Tomáš	Starý	2	25.02.2018
1	Tomáš	Starý	6	15.03.2018
2	Josef	Barevný	1	22.01.2018
2	Josef	Barevný	3	13.01.2018
3	Lucie	Strakatá	4	25.02.2018
3	Lucie	Strakatá	5	25.01.2018
3	Lucie	Strakatá	7	22.04.2018
4	Petra	Jedlá	NULL	NULL

U pravého spojení je to obdobné, jenom hodnoty NULL bychom viděli na opačné straně a úplné spojení je pak kombinace levého a pravého spojení.

Specifikované spojení s klíčovým slovem ON

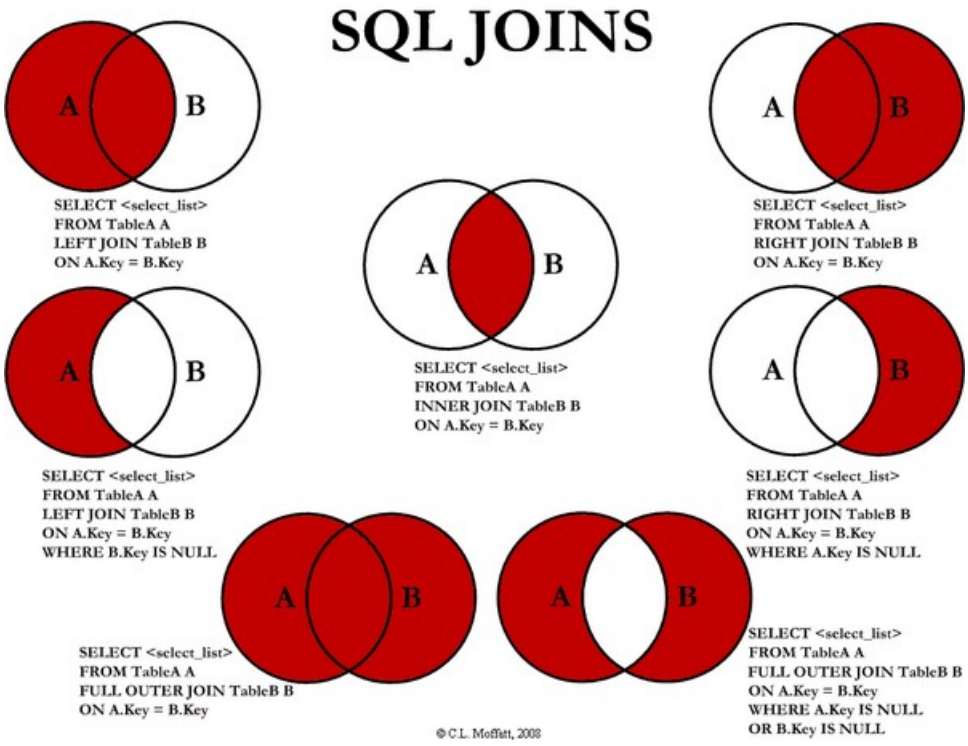
Tohle spojení využívá predikáty vyskytující se za klíčovým slovem ON a spojí právě jen ty řádky, které odpovídají daným predikátům, a proto vidíme jen ty řádky, které mají odpovídající protějšky.

```
zamestnanec z JOIN dochazka d ON z.id = d.cislo
```

Tabulka 3: Specifikované spojení s ON

Z.id	Z.jmeno	Z.prijmeni	D.cislo	D.datum
2	Josef	Barevný	1	22.01.2018
1	Tomáš	Starý	2	25.02.2018
2	Josef	Barevný	3	13.01.2018
3	Lucie	Strakatá	4	25.02.2018
3	Lucie	Strakatá	5	25.01.2018
1	Tomáš	Starý	6	15.03.2018
3	Lucie	Strakatá	7	22.04.2018

Pro další vizuální ilustraci spojení si můžeme podívat (viz obr odkaz), kde vidíme pěkné a jasné vyobrazení různých spojení pomocí množin A a B, červené vyobrazené části jsou výsledky spojení. [6]



Obrázek 3: Různé typy spojení [7]

3.7 Klauzule WHERE

Tahle klauzule slouží jako filtr řádků. Obsahuje predikáty, které nám zmenšují výslednou množinu dat. Výsledkem predikátu je TRUE nebo FALSE. Když se porovnává hodnota NULL s kteroukoliv jinou hodnotou, výsledkem je vždy UKNOWN, což je pouze označení pro hodnotu, která nemá žádnou hodnotu a je jenom označení, že hodnota nějakého datového typu je prázdná.

Ostatní hodnoty se porovnávají dle posloupnosti (znakové řetězce), číselného pořadí (číselné typy), velikosti (intervalové hodnoty) nebo chronologického pořadí (datum, čas). K porovnání se například používají operátory: =,<,>,<=,>=,<> (nerovná se). Operátory jako: || (zřetězení) nebo * (násobení) můžeme použít, jenom pokud to daný datový typ podporuje. Dále můžeme používat buď vestavěné funkce, nebo námi vytvořené funkce, které vrací danou hodnotu, kterou následně porovnáme a dostaneme výsledek TRUE či FALSE. Při výskytu více predikátů se používají logické operátory (AND a OR), které nám sdružují jednotlivé predikáty a určují výsledek toho, zda daný řádek vyhovuje všem predikátů či nikoliv.

Můžeme si popsat pár predikátů:

- LIKE

  ```
  B LIKE 'řetězec'
  ```

  Predikát předpokládá, že B je datový typ řetězec a hledá specifikovaná podřetězec. Dají se v něm používat různé znaky ([]%_), které určují specifickou vlastnost, kterou musí daný řetězec splňovat.

- IN

  ```
  B IN (X, Y, Z,...)
  ```

  V tomto případě se jedná o porovnání hodnoty B s výčtem hodnot, a pokud se v něm nachází shodná hodnota, pak se vrací TRUE, jinak FALSE. Samozřejmě, že výčet hodnot může být tvořen vnořeným dotazem, který nám stejně ve výsledku dává výčet hodnot, se kterým danou hodnotu porovnáváme.

- BETWEEN

  ```
  X BETWEEN A AND B
  ```

  Predikát odpovídá výrazu (A <= X) AND (X <= B). A i B musejí být ve vzestupném pořadí. X BETWEEN B AND A by se prováděl jako (B <= X) AND (X <= A) a byl by FALSE, pokud by nebyly všechny hodnoty stejné.

- IS NULL

```
B IS NULL
```

  Tento predikát slouží k testování NULL. OD jiných predikátů se liší v tom, že jeho výsledek může být pouze TRUE nebo FALSE – nikoliv UNKNOWN. [6]

3.8 Poddotazy

Dotazy jsou příkazy SELECT, které odvozují či extrahují data z tabulek nebo pohledů. Poddotazy, jinak řečeno vnořené dotazy, jsou dotazy, které produkují takový data, která nejsou finálním výstupem příkazu, ale budou dále zpracovávána příkazem.

Poddotazy se mohou vyskytovat v predikátech jiných dotazů. Můžeme je najít i v příkazech UPDATE nebo DELETE, v klauzulích FROM nebo ve výrazech. Vždy, kdy je použijeme, musejí být uzavřené v závorkách. Skutečnost, že poddotazy mohou být použité i v jiných predikátech, znamená, že je můžeme dávat i do jiných poddotazů a to do libovolné úrovně zanoření. Existují tři typy poddotazů:

- **Tabulkové poddotazy** produkují libovolný počet řádků. Často se používají ve FROM klauzuli jako vnořená tabulka nebo v predikátu EXISTS.
- **Jednořádkové poddotazy** produkují nejvýše jeden řádek, ale mohou obsahovat libovolný počet sloupců. Ty se třeba uvádějí v konstruktorech hodnot řádků.
- **Skalární poddotazy** mohou vytvořit nejvýše jednu hodnotu. Můžeme je vidět například v klauzuli SELECT, kde se vyžaduje jen jedna hodnota ve sloupci pro určitý řádek.

Další možné dělení poddotazů je:

- **Závislý (korelovaný) poddotaz**, který je, jak vidíme v názvu, závislý na jinem dotazu. Nedá se provést samostatně. Korelovaný poddotaz se vyhodnocuje opakovaně pro každý řádek vnějšího dotazu.
- **Nezávislý (nekorelovaný) poddotaz** není závislý na žádném jiném dotazu a vyhodnocuje se jen jednou.

```
SELECT *
FROM Zamestnanec z1
WHERE plat >= (SELECT AVG(plat)
           FROM Zamestnanec z2
           WHERE z1.Oddeleni = z2.Oddeleni)
```

Výpis 3: Ukázka závisleho poddotazu

Predikáty ANY a ALL

Tyhle predikáty pracují výhradně s poddotazy. ANY a jeho synonymum SOME fungují obdobně jako IN. Tedy procházejí se všechny vrácené hodnoty z poddotazu a postupné se porovnávají s danou hodnotou.

Jelikož IN a =ANY jsou ekvivaletní, tak bzchom si mohli myslet, že NOT IN a <>ANY jsou taktéž ekvivaltní, ale není tomu tak. Ekvivalentem NOT IN je <>ALL. Kdybychom se nad tím zamysleli, zjistili bychom, že je to zcela logické. <>ANY vrátí hodnotu TRUE pokaždé, kdy poddotaz vrátí více než jednu hodnotu. Když poddotaz vrátí dvě a více různých hodnot, tak vždy bude mezi nimi jedna, která nebude odpovídat zadané skalární hodnotě. Naproti tomu <>ALL funguje přesně jako NOT IN. Vrací hodnotu TRUE pouze tehdy, pokud žádná hodnota vrácená poddotazem není rovna hledané skalární hodnotě.

Z tohoto vyplývá fakt, že ALL se využívá častěji s operátorem nerovnosti (<>) než s operátorem rovnosti (=). Mějme tyhle dva dotazy:

```
SELECT *
FROM Zbozi
WHERE id = ANY (SELECT id FROM Zbozi WHERE typ = 'Elektronika')
```

Výpis 4: Dotaz požívající predikát ANY

```
SELECT *
FROM Zbozi
WHERE id = ALL (SELECT id FROM Zbozi WHERE typ = 'Elektronika')
```

Výpis 5: Dotaz požívající predikát ALL

U prvního vidíme použití predikátu ANY, v tomhle případě dostaneme takové záznamy zboží, jejichž id vyhovují hodnotám vraceným z vnořeného dotazu, ale pak si musíme dát pozor u druhého dotazu, že tam dostaneme jen takové zboží, které odpovídá všem vráceným hodnotám z poddotazu. Proto si musíme dávat pozor, kdy použít ANY nebo ALL.

Predikát EXISTS

EXISTS je predikační funkce, která přijímá jako svůj parametr pouze poddotaz. Pracuje na jednoduchém principu, jestliže poddotaz vrátí nějakou výslednou množinu

(jakoukoliv), vrací výsledek TRUE, jinak FALSE. Poddotaz musí být ohraničený závorkami.

Poddotaz předaný jako parametr je většinou korelační poddotaz, ale můžeme se setkat i s nezávislým poddotazem. V poddotazech bychom měli pravidelně používat příkaz SELECT *. To pomáhá optimalizátoru si vybrat použitý sloupec, což zajišťuje vyšší výkon.

Existuje vztah mezi predikáty EXISTS a IN a to takový, že se dá lehce převést jeden na ten druhý. Ukážeme si příklad, kdy dotaz, kde se používá predikát EXISTS převede na dotaz s použitím IN:

```
SELECT * FROM Zakaznik z
WHERE EXISTS(SELECT *
         FROM Ucet u
         WHERE z.IDzakaznik = u.IDzakaznik and u.datumVytvoreni <= '
            01.01.2018')
```

Výpis 6: Poddotaz zapsaný pomocí prdikátu EXISTS

```
SELECT * FROM Zakaznik z
WHERE z.IDzakaznik IN (SELECT u.IDzakaznik
                       FROM Ucet u
                    WHERE u.datumVytvoreni <= '01.01.2018')
```

Výpis 7: Poddotaz zapsaný pomocí prdikátu IN

U obou dvou dotazů dostaneme shodný výsledek (zákazníky, jejichž účet byl vytvořen po datu 1. ledna 2018). Občas se takové transformace hodí, pokud se například chceme vyvarovat korelačním poddotazům. [8]

3.9 Klauzule GROUP BY

Touhle klauzulí se definují skupiny výstupních záznamů, na které se aplikují nějaké agregační funkce (SUM, COUNT, MAX atd.). Pokud jsou v klauzuli SELECT uvedené nějaké názvy sloupců, které nejsou volané agregační funkcí, tak musí být uvedené i v klauzuli GROUP BY, podle nich se budou seskupovat data.

Všechny výstupní řádky, získané z dotazu, které mají stejnou hodnotu ve všech sloupcích, podle nichž se seskupuje, vytvoří skupinu (pro účely GROUP BZ se všechny hodnoty NULL považují za stejné). Na každou takovou skupinu se bude aplikovat agregační funkce.

3.10 Klauzule HAVING

Má stejnou funkci jako klauzule WHERE, filtruje záznamy. Její největším rozdíle od klauzule WHERE je, že nefiltruje záznamy získané z kartézského součinu, ale ze skupin řádků, získané z klauzule GROUP BY. Filtruje tedy skupiny na základě výsledných hodnot agregačních funkcích. Proto nemůžeme agregační funkce (jako predikát) používat v klauzuli WHERE, ale jenom v klauzuli HAVING pro to určené.

```
SELECT IDuctenka, AVG(castkaKZaplaceni)
FROM Uctenka
GROUP BY IDuctenka
HAVING AVG(castkaKZaplaceni) > 9000
```

Výpis 8: Ukázka použití klauzulí GROUP BY a HAVING

V předchozím příkazu dostaneme jako výsledek účtenky, jejichž částka k zaplacení je větší než průměrná částka všech účtenek. Dále bych chtěl jen znovu zdůraznit, že predikát obsahující agregační funkci nemůžeme dát do WHERE klauzule, jinak bychom dostali chybu a dotaz by se neprovedl.

3.11 Klauzule ORDER BY

Klauzulí ORDER BY se výstup seřadí. Řádky se řadí dle hodnot ve sloupcích, které jsme uvedli v této klauzuli. Největší prioritu má první sloupec, další sloupec určuje pořadí v rozsahu duplicitních hodnot prvního sloupce atd. Defaultně je nastavené vzestupné řazení, ale můžeme ho přenastavit pomocí klíčových slov ASC (defaultní) a znamená vzestupné řazení nebo DESC, to nastaví sestupné řazení. Místo názvů sloupců můžeme dat čísla, která budou odpovídat pořadovým číslům (umístění) sloupců uvedených v klauzuli SELECT.

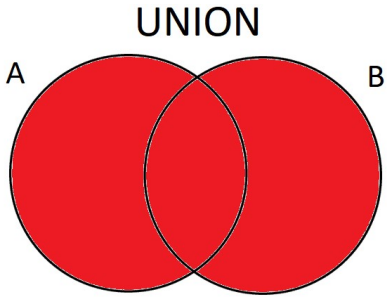
3.12 UNION, INTERSECT a EXCEPT

Tyto klauzule přijímají jako argument celé dotazy, ale nesmí obsahovat klauzuli ORDER BY, ve tvaru:

```
dotaz A { UNION | INTERSECT | EXCEPT } [ALL] dotaz B
```

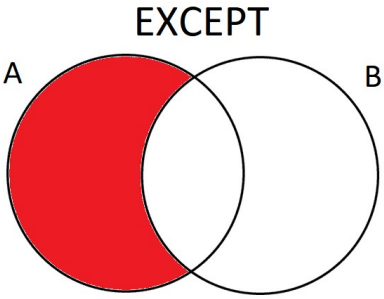
Všechny výstupní sloupce všech dotazů musí být stejné (počet i názvy), protože se porovnávají ve specifickém pořadí. Dále si vysvětlíme, co se přesně znamenají výše zmíněné operace. Mějme dvě množiny A a B, které budou představovat množinu výstupních řádků pro dotaz A a B, pak:

- **UNION** vytvoří sjednocení obou množin a výsledek je tedy obsah obou dvou množin. Při použití **UNION ALL** docílíme toho, že se nebudou mazat duplicitní řádky.



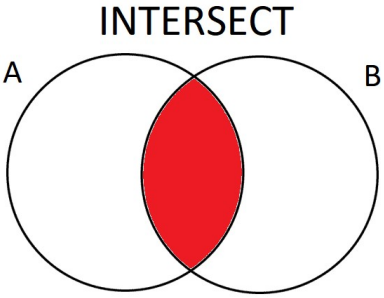
Obrázek 4: Vizuální ukázka sjednocení dvou množin

- **EXCEPT** představuje tu část množiny A, která nemá nic společného s množinou B. Výsledkem jsou tedy záznamy z množiny A, které se ještě nevyskytují v množině B. Normálně se pracuje s unikátními řádky a duplicity se mažou, ale pokud uvedeme **EXCEPT ALL**, tak se budou do výsledku zahrnovat i duplicity, tedy duplicity se nebudou mazat.



Obrázek 5: Vizuální ukázka rozdílu množin A - B

- **INTERSECT** vrací průnik obou množin. Výsledkem jsou tedy řádky, které se nacházejí ve výsledcích obou dvou dotazů. Pokud opět uvedeme **INTERSECT ALL**, tak se nám nebudou eliminovat duplicitní řádky. [6]



Obrázek 6: Vizuální ukázka průniku dvou množin

4 Zmenšení databaze

Mějme databázi s x tabulkami a každá tabulka má n řádků (počet záznamů). Když těch záznamů je velké množství a chceme danou databázi zmenšit, pak na to můžeme použít nějaký nástroj na to určený.

Jinak řečeno, zmenšení databáze je proces, při kterém vybíráme nějakou množinu záznamů (menší množina než ta původní) z databáze, kterou pak například můžeme vložit do nové databáze, a tudíž získáme menší databázi jak vzhledem k počtu záznamů, tak i velikosti na disku.

Způsoby zmenšení databáze mohou být různé. Můžeme zmenšit databázi na základě celkového počtu záznamů, přičemž požadujeme, aby se databáze například zmenšila na 25% původní velikosti, nebo dalším řešením a asi vhodnějším pro nás je, zmenšení na základě použiti SQL dotazů a především WHERE klauzulím, pomocí kterých dostaneme vybranou množinu dat, která nám může tvořit data (množinu záznamů) v zmenšené databázi.

4.1 Nástroje pro zmenšení databáze

4.1.1 Toad for Oracle

Tenhle nástroj je určený pro práci především s Oracle databázemi. Nástroj nabízí bohatou škálu možností, se kterou můžeme pracovat s databázemi. Šetří čas a snižuje rizika při vývoji a údržbě Oracle databází.

Tenhle nástroj umožnuje:

- spouštět různé diagnostiky
- provádět různé operace (například CRUD operace)
- vytvářet různé skripty
- umožnuje administraci databáze
- upravovat databázi
- zmenšení databáze
- import/export dat
- a mnoho dalších věcí [9]

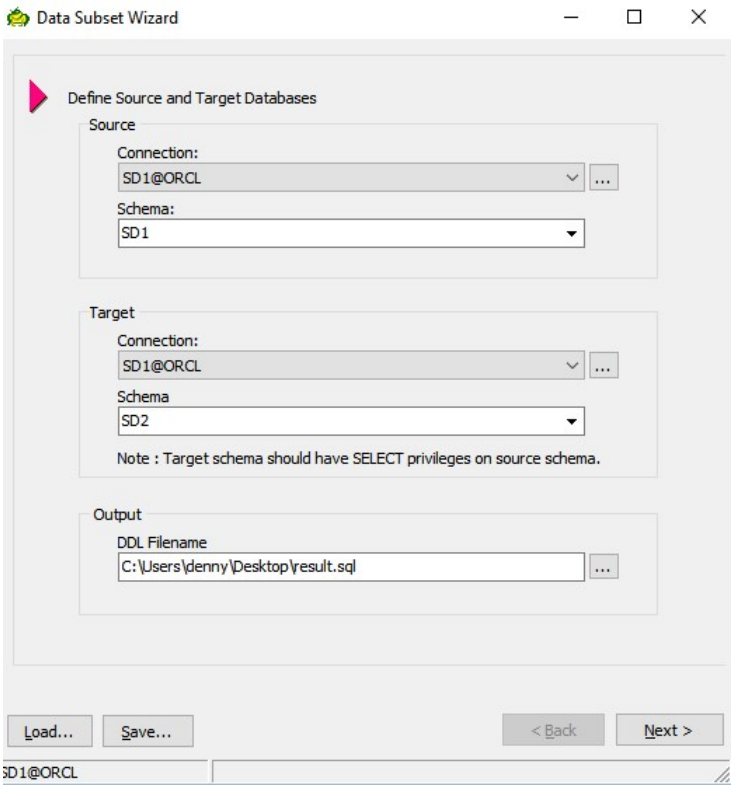
4.1.1.1 Zmenšení databáze v Toad for Oracle

Jak už bylo zmíněno, Toad umí i zmenšit databázi. Tuhle možnost program nabízí, když si rozklikneme na horní liště Database, pak Export a pokračujeme vybráním

možnosti Data Subset Wizard. Tím se dostaneme do nového okna, kde po vyplnění potřebných informací:

- zdrojové a cílové schéma
- umístění výsledného skriptu
- možnost přidat do skriptu vytvoření objektů (tabulka, index, pohled...)
- množství dat v procentech
- a dalších

Následně po projití celého „wizardu“ dostaneme skript, který nám zmenší naši databázi.



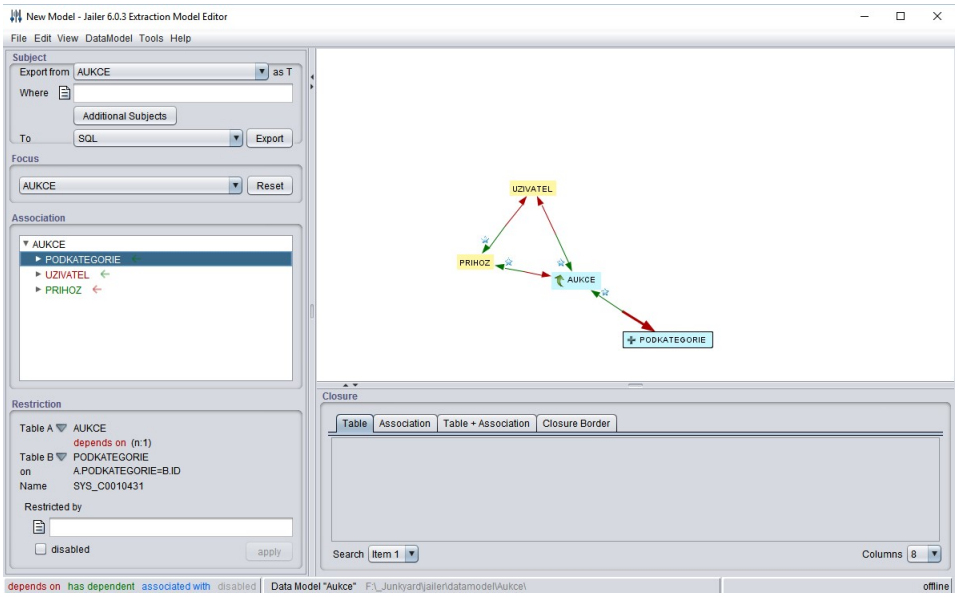
Obrázek 7: Data subset wizard

4.1.2 Jailer

Jailer je nástroj pro zmenšení databáze, zobrazení schéma databáze a zobrazení dat. Exportuje data z relační databáze a uloží je jako SQL skript. Importuje data do

tvého vývojového a testového prostředí. Umí vymazat zastaralá data bez narušení integračního omezení. Je nezávislý na platformě. Generuje datové sady DbUnit, hierarchicky strukturované XML a topologicky setříděné SQL-DML.

Navíc je to open source, je zcela napsaný v javě. K připojení k databázi je nutné použít API (například JDBC). [10]



Obrázek 8: Ukázka programu Jailer

4.1.2.1 Zmenšení databáze v Jailer

Jailer nám umožnuje zmenšit databázi pomocí klauzulí WHERE. Nejdříve si musíme vybrat hlavní tabulku, dle které se bude zmenšení odvíjet. Tedy vybereme si např. tabulku: Uživatel a k ním si napíšeme klauzuli WHERE (podmínku): id <= 100. Dále klikneme na export, a tam už jen doplníme informace (např. umístění skriptu, zdrojové a cílové schéma) a následně generujeme skript.

Celý proces zmenšení začíná námi vybrané tabulce. Pak z tabulky dostaneme menší množinu dat, které splňují námi vloženou podmínku, a pak se pokračuje dle referencí na další tabulky a v nich se zachovávají data, která mají referenci na již zmenšenou tabulku.

Kdybychom měli tabulku uživatel a po provedení dotazu s danou podmínkou dostali jen záznamy s id: 1, 2 a 3, tak v tabulkách, které mají s ní referenci, by si nechali jen ty záznamy, které mají referenci k záznamům v tabulce uživatel, a takhle se pokračuje na další tabulky, které mají reference na již zmenšené tabulky.

4.1.3 Databee

Nástroj pro zmenšení databáze. Pracuje jak s Oracle tak i SQL Server databázemi. DataBee se skládá ze tří aplikací, každá z nich je zaměřena na určitou funkcionalitu. Takové rozdělení nám ulehčí použití softwaru.

4.1.3.1 Set Designer

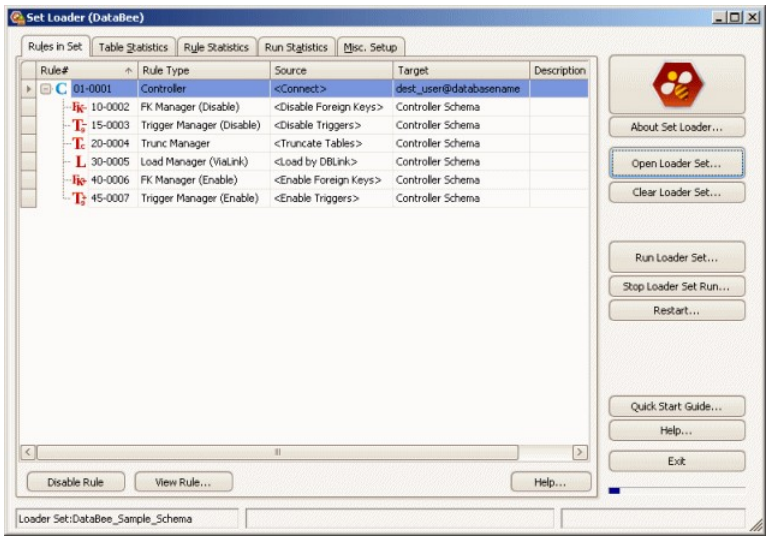
Slouží především k zjištění, jaké objekty databáze obsahuje. Indikuje, jak jsou tabulky mezi sebou propojené. Obsahuje nástroj, který pomáhá sestavit kolekci obsahující informace o připojení, pravidla a informace o tabulkách vyžadovaná pro extrakci (Extraction Set) a pro načítání operací (Loader Set). Navíc může zjistit a automaticky přidat pravidla pro objekt a vztahy vyskytující se v databázi či editaci těchto pravidel.

4.1.3.2 Set Extractor

V podstatě se jedná o „ROWID Hunter“. To co dělá, je identifikace kolekce řádků (dle ROWID) ve zdrojovém schématu. Výstupem je list obsahující „ROWID hodnoty“ v dočasné tabulce uložené ve zdrojovém schématu.

4.1.3.3 Set Loader

Používá Loader Set k načtení řádků extrahovaných pomocí aplikace Set Extractor do cílového schématu. Během extrakce ROWID (čísla řádků) jsou uložena v dočasné tabulce ve zdrojovém schématu. Pak se připojí do cílového schématu a přistupuje k těm extrahovaným ROWID (číslům řádků) a použije je k načtení cílového schématu v databázi. [11]



Obrázek 9: Ukázka Aplikace Set Loader

4.2 Proč implementovat svůj vlastní nástroj?

Všechny zmíněné programy nám sice poskytují možnost zmenšení databáze, ale vždy jsme omezeni možnostmi, podle kterých databázi zmenšíme, například v nástroji Toad for Oracle můžeme zmenšit jen Oracle databáze a navíc nám nabízí ji zmenšit způsobem, že si nastavíme, na kolik procent z původních dat ji chceme zmenšit, a nenabízí nám možnost přidat sadu SQL dotazů, které by bral v potaz při zmenšování databáze.

Na druhou stranu nástroj Jailer už není závislý na platformě a dokonce umí zmenšit databázi s ohledem na SQL dotazy, ale limitovaný tím, že se vždy musí nastavit hlavní tabulka, od které se začne proces zmenšování, navíc je nutné ručně nastavovat podmínky pro všechny tabulky a v případě velkého počtu tabulek nám to bude nesmírně dlouhou dobu. Takových detailů bychom mohli nalézt více.

V mém případě potřebuji takový nástroj, který vezme v potaz, že mám sadu SQL dotazů, které si jednoduše načtu, a pak následně vygeneruji skript, který mi danou databázi zmenší.

5 Implementovaná aplikace

V implementační části si důkladně popíšeme implementovanou aplikaci z hlediska její struktury a funkčnosti. Následně se pobavíme, jak aplikace vlastně provádí svůj úkon, neboli si objasníme algoritmus, který se používá pro zmenšení databáze na základě vytížení.

5.1 Základní charakteristika

Aplikace se nazývá Db subsetter 1.0. Aplikace byla navržena a vytvořena pomocí nástroje MS Visual Studia 2015 v jazyce C#. Aplikace je spustitelná v operačních systémech Windows od Společnosti Microsoft s přítomnosti .NET frameworkem 4.6.1 a vyšším. Při implementaci aplikace byla využita MVC.

5.2 Vstupní požadavky

Očekávané vstupy naší aplikace jsou následující:

1. soubor se vstupními dotazy

   Tenhle soubor musí mít příponu .txt nebo .sql. Dále musí obsahovat dotazy, které jsou spustitelné nad zdrojovou databází. Pokud se v souboru nachází více vstupních dotazů, tak musí být oddělené znakem „;“ (středník).

2. dvě databáze (zdrojovou a cílovou)

   Ke správnému provedení operace zmenšení jsou vyžadovány dvě databáze, tudíž když máme *zdrojovou databázi*, ze které budeme vycházet, tak musíme mít, popřípadě vytvořit, i *cílovou databázi*. Aplikace nevytváří žádnou jinou databázi, spoléhá na to, že už cílová databáze existuje a do ní jen převede výsledek zmenšení.

   Důležitá zmínka je, že obě tyto databáze musí být na stejné instanci (stejný databázový server). Jak už bylo zmíněno dříve, tak obě databáze musí být typu MS SQL Server. Tohle všechno je z důvodu toho, aby implementace zmenšení nebyla příliš obtížná a zbytečně komplikovaná tím, že bychom museli řešit kopírování dat (i obrovských dat) mezi různými instancemi SQL Serveru. Tím, že máme obě databáze na stejném serveru, ke kopírování tabulek či dat nám stačí *kolekce příkazů insert-select*.

5.3 Struktura

Aplikace je vytvořena pomocí několika vrstev, které mezi sebou komunikují. Jsou sice na sobě závislé, ale jsou lehce nahraditelné, upravovatelné a dají se lehce rozšířit. Jedná se o vrstvy:

- **Datová**, která má přímý přístup k databázi. Komunikuje s ní a provádí nad ní určité operace.
- **Logická**, ta zajištuje tzv. logiku programu, kde zpracovává požadavky prezentační vrstvy, provádí různé výpočty, v našem případě se v této vrstvě provádí samotný algoritmus zmenšení databáze, skenuje se vstupní databáze či posílání různých požadavku do databáze skrze datovou vrstvu.
- **Prezentační**, jenž její nejhlavnější úkol je komunikace s uživatelem. V podstatě se jedna o GUI (grafické rozhraní), pomocí kterého můžeme například nastavit a následně provést zmenšení databáze.

Dále si popíšeme, z jakých části se aplikace skládá a co každá část dělá a za co je zodpovědná a proč je to tak rozdělené. Aplikace se tedy skládá z:

- knihovny pro zpracování SQL dotazů,
- knihovny pro přístup k databázi,
- knihovny pro zmenšení databáze
- a GUI

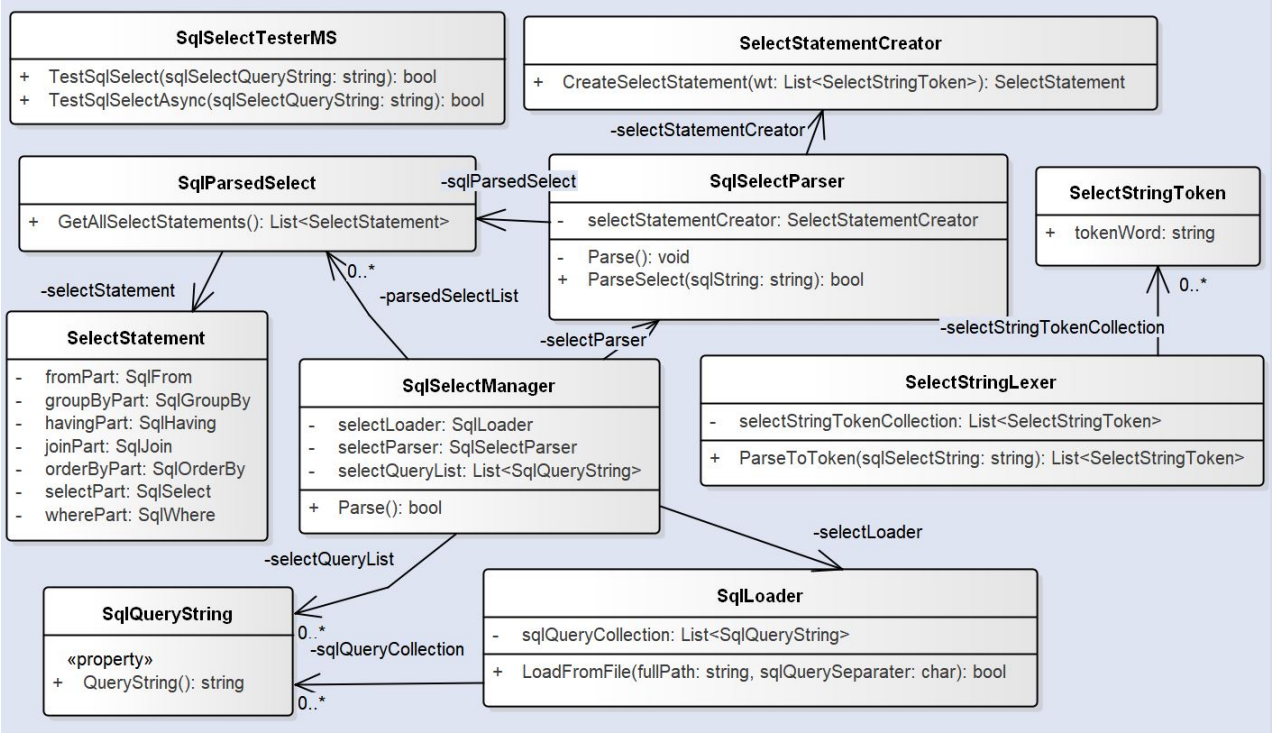
5.3.1 Knihovna pro zpracování SQL dotazů

Tahle knihovna slouží pouze pro zpracování SQL dotazů napsaných pro MS SQL Server. Tuhle knihovnu můžeme zařadit do logické vrstvy. Zpracováním dotazů v našem případě myslím parsování SQL dotazů, neboli česky řečeno: rozklad dotazu na jednotlivé části (klauzule) pro pozdější zpracování. Účelem této knihovny je tedy převést řetězcovou reprezentaci vstupního SQL dotazu na objektovou. Neumí zpracovat žádné jiné SQL operace jako: vložení, smazání nebo upravování. Aplikace se zaměřuje pouze na SQL Select a zpracování dotazů má i svých pár omezení a nevýhod, ale o tom si pak řekneme v 5.3.1.3.

Určitě se můžete ptát, zda už neexistuje nějaká jiná knihovna či nástroj na tuto problematiku a proč ji tedy nepoužít, o tom si pobavíme v následující kapitole 5.3.1.4.

5.3.1.1 Struktura

Knihovna se skládá z několika částí, které jsou důležité pro to, aby se daný dotaz správně zpracoval. Na obrázku 10 vidíme třídní diagram, ve kterém se nacházejí pouze ty nejdůležitější třídy, metody s atributy.



Obrázek 10: Třídní diagram knihovny pro zpracování SQL dotazů

Názvy částí se shodují s názvy tříd nacházející se v této knihovně:

- **SQL Loader**

  Jeho hlavním funkcí je načíst data (dotazy) ze souboru do listu sqlQueryCollection. Jelikož dotazy se ze začátku nacházejí v souboru, je tedy nutné je nahrát a uložit si je do našich předem připravených objektů, které pak budeme dále potřebovat pro další zpracování.

  Dokáže při čtení řádku v souboru eliminovat řádky začínající znaky „–“, které představují v SQL jazyce řádkové komentáře, a které byly nepodstatné pro zpracování.

  Další věcí, kterou si musíme uvědomit, je, že SQL Loader rozděluje načtené dotazy ze souboru dle specifického znaku „;“ (středník). Tento požadavek je nad rámec nutné syntaxe v T-SQL, ale nutný pro správné načtení dotazů.

- **SelectStringLexer**

  Tato třída je určená pro lexikální analýzu řetězců vstupních SQL dotazů. Načte si řetězec a po té ho rozdělí na množinu tokenů, které vrátí v listu. Řetězec se rozděluje dle mezer.

- **SelectStatementCreator**

Instance této třídy zpracovává list tokenů získaných ze SelectStringLexeru. Zpracuje je a vytvoří z nich objekt typu SelectStatement.

Tokeny se zpracovávají tak, že se postupně čtou a dle klíčových slov (SELECT, FROM či WHERE) se rozdělují do jednotlivých části dotazu. Výsledný objekt se tedy skládá z částí, které jsou pojmenované dle klíčových slov, které najdeme v běžném SQL dotazu:

- klauzule **SELECT** - Obsahuje list sloupců.
- klauzule **FROM** - Obsahuje list tabulek vyskytujících se pouze v daném (vnějším) dotazu nikoliv ve vnořených dotazech.
- klauzule **JOIN** - Obsahuje informace o spojení v dotazu:
  - typ spojení
  - levá a pravá tabulka
  - podmínka spojení
- klauzule **WHERE** - Obsahuje list podmínek, dle kterých se filtrují data. Tyhle podmínky jsou pak dále potřeba pro zmenšení databáze.
- klauzule **GROUP BY** – Obsahuje list sloupců, dle kterých se seskupují data
- klauzule **HAVING** – Obsahuje podmínky, které filtrují skupiny vytvořené dle předchozí klauzule.
- klauzule **ORDER BY** – Obsahuje list sloupců, dle kterých se bude výsledek seřazovat.

První token musí být vždy slovo SELECT, tím se potvrdí, že se jedná o SQL příkaz typu Select. Pokud se tedy narazí na další klíčové slovo SELECT, tak dojde k rekurzi, kde se načtou všechny tokeny od závorky před klíčovým slovem SELECT až do tokenu, který znamená konec tohohle vnořeného dotazu (poddotazu). Konec je určen dle toho, že každý vnořený dotaz je ohraničen kulatými závorkami. Dochází tedy k počítání levých a pravých závorek. Na začátku je jedna levá závorka v listu závorek, když se najde další levá závorka, tak se přidá do tohoto listu a když se najde pravá závorka, tak se odebere poslední přidaná levá závorka z listu. Pokračuje se tak dlouho, dokud list závorek není prázdný. Tím dostaneme vnořený dotaz, kterým opět zpracováváme jako ten vnější dotaz.

- **SqlSelectParser**

  Tahle třída se tváří jako parser. Obsahuje v sobě SelectStringLexer a SelectStatementCreator, pomocí kterých zpracuje vstupní řetězec SQL dotazu. Jako výsledek vytvoří objekt typu SqlParsedSelect, který obsahuje SelectStatement (rozložený řetězec na části) a originální řetezec SQL dotazu.

- **SqlSelectManager**

  Tenhle manažer obsahuje všechny výše popsané funkcionality. Tedy stačí mu dát na vstup soubor s SQL dotazy a vrátí nám list zpracovaných dotazů.

  Rozdíl mezi SqlSelectParser a SqlSelectManager je ten, že parser slouží pro zpracování jednoho řetězce a manager obsahuje parser a další funkce. Může například zpracovat celý list řetězců nebo je jen načíst ze souboru do paměti.

- **SqlSelectTester**

  Testuje dotazy, zda jsou spustitelné, v našem případě, nad zdrojovou databází. Tohle je velmi důležité, protože pokud by nebyly dotazy spustitelné, pak by mohla nastat situace, že i samotný proces zmenšení by nebyl úspěšný. Tuhle funkci můžeme využít například v GUI.

5.3.1.2 Vstupy a výstupy

Jediný vstup, který tato knihovna bere, je soubor obsahující vstupní SQL dotazy. Všechny tyto dotazy musí být syntakticky správně napsané a spustitelné na zdrojovou databází. V dotazech nesmí obsahovat blokové komentáře (/* */) z důvodu toho, že by se potom takový dotaz špatně zpracoval.

Výstup této knihovny je list objektů, které obsahují zpracované řetězce na části, ke kterým můžeme jednoduše přistupovat.

5.3.1.3 Omezení

Nyní si popíšeme omezení naší knihovny pro zpracování SQL příkazů, jelikož není schopna zpracovat libovolný SQL dotaz. Tahle knihovna umí zpracovat dotazy, které:

1. začínají klíčovým slovem SELECT
2. obsahují klauzule:
   - povinné: SELECT, FROM
   - nepovinné: JOIN, WHERE, GROUP BY, HAVING a ORDER BY
3. obsahují predikáty: IN, ANY, ALL, EXISTS, BETWEEN, NOT, LIKE, CASE
4. obsahují logické operátory: OR, AND
5. obsahují agregační funkce: SUM, COUNT, AGR
6. obsahují unární funkce (funkce s jedním parametrem)
7. obsahují klíčové slovo DISTINCT pouze za klíčovým slovem SELECT obsahují vnořené dotazy, které rovněž splňují všechny výše splněné podmínky, ve všech klauzulích zmíněných v bodě 2

Knihovna neumí zpracovat dotazy, které:

1. mají podmínky, kde je hodnota na pravé straně (např.: 'Nina' = jmeno)
2. obsahují takové predikáty, které nejsou zmíněné v bodě 3
3. obsahují funkce se dvěma a více parametry
4. obsahuje příliš zbytečných závorek např.: SELECT * (FROM) TABULKA (t) WHERE ((podmínka))
5. obsahující všechny množinové operátory (UNION, INTERSECT,...)

Omezení si můžeme popsat pomoci gramatiky:

```
příkaz select = SELECT [DISTINCT]
{ [prefix.] { * | název sloupce }
| příkaz select
| agregační funkce } [AS] [alias] [ ,...n ]
FROM { název tabulky [AS] [alias] | příkaz select [AS] alias }
[ [LEFT | RIGHT | FULL] [INNER | OUTER] JOIN název tabulky [AS] [alias]
ON { podmínka | příkaz select } [ AND | OR ...n ] ]
[WHERE [NOT] { podmínka | příkaz select }] [ AND | OR ...n ]
[GROUP BY [prefix.] název sloupce [ ,...n ] ]
[HAVING agregační funkce { = | != | < | > | <= | >= }
{vyraz | příkaz select} [ AND | OR ...n ] ]
[ORDER BY [prefix.] název sloupce [ASC | DESC] [ ,...n ] ]

podmínka = { [prefix.] název sloupce
{ { = | != | < | > | <= | >= } | predikát } { vyraz | příkaz select }
| EXISTS příkaz select }

predikát = { IN | ALL | ANY | BETWEEN | LIKE | CASE }
```

5.3.1.4 Proč vlastní knihovnu

Důvodem, proč byla implementována vlastní knihovna na zpracování SQL dotazů, je ten, že při integraci již existujících knihoven docházelo buď k problémům při kompilaci, nebo byly napsány v jiných jazycích, které nepodporují snadnou integraci do .NET frameworku.

Byla například vyzkoušena knihovna s názvem General SQL parser. Tato knihovna je příliš obecná a zároveň placená. Rovněž pasuje do .NET frameworku. Při testech docházelo k problémům propojení této knihovny s naší aplikací. Po dlouhé analýze a pokusech tuto knihovnu integrovat, jsme došli k názoru, že si vytvoříme vlastní.

5.3.2 Knihovna pro přístup k databázi

Tahle knihovna spadá do části datové vrstvy. Můžeme ji chápat jako prostředníka mezi logickou vrstvou a samotnou databází. Proto její hlavní funkcí je komunikace s databází a nesmíme zapomenout, že databáze musí být na MS SQL serveru.

5.3.2.1 Struktura

Tato knihovna se skládá z částí (názvy částí jsou odvozené ze jmen tříd):

- **DatabaseMS**

  Představuje „databázi“, ke které poskytuje připojení a další informace či operace jako například:

  - stav či otestovat připojení
  - možnost vytvořit transakci

- **DbManagerMS**

  Je to jakýsi manažer, který poskytuje komunikaci s databází. Přes něj můžeme volat určité příkazy do databáze, a tudíž provádět operace či dostávat nějaké výsledky z databáze.

- **DbRepositoryMS**

  Jedná se o repositář, který obsahuje všechny příkazy, se kterými „komunikujeme“ s databází. Jedná se třeba o: vkládání či kopírovaní dat, tvorba tabulek či schémat nebo získání dat z různých dotazů.

- **DbTable**

  Objekt, který představuje tabulku v databázi a obsahuje o ní informace: jméno tabulky, schématu a databáze a list sloupců.

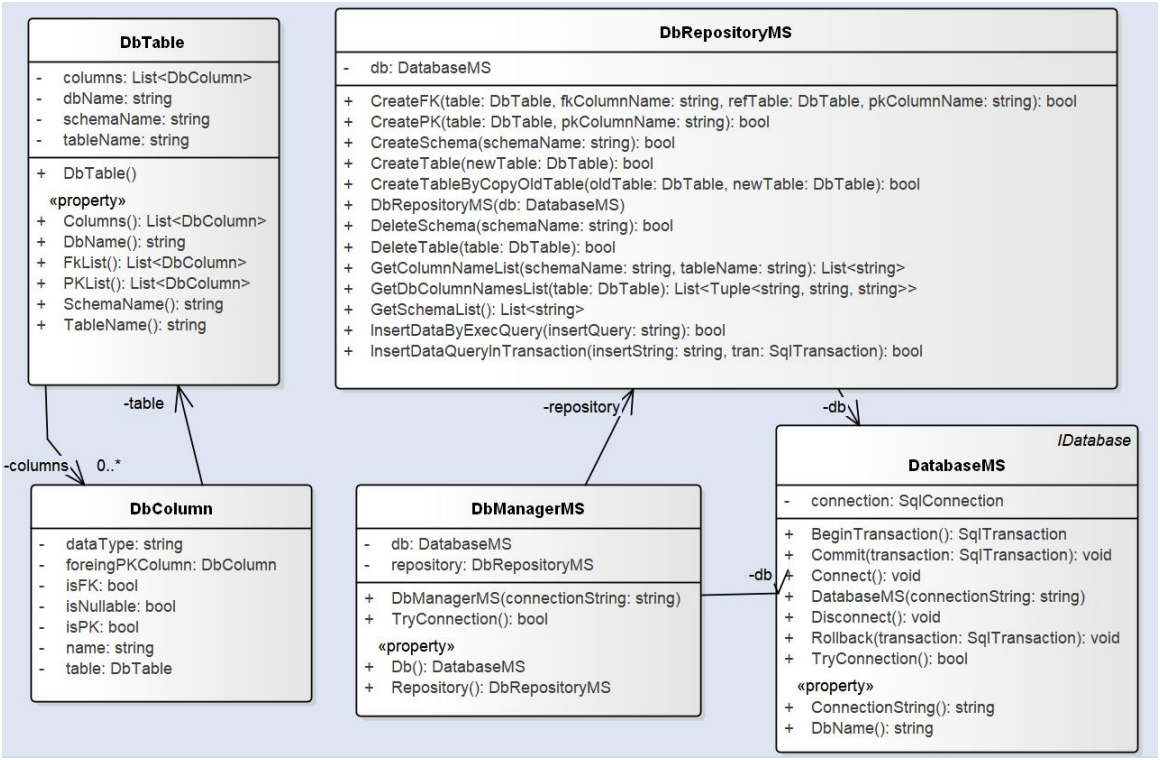
- **DbColumn**

  Objekt, který představuje jeden sloupec v tabulce. Obsahuje například název sloupce nebo datový typ (Viz 11).

5.3.3 Knihovna pro zmenšení databáze

Tahle knihovna je z pohledu funkcionality nejdůležitější část programu, protože obstarává podstatnou část celého procesu zmenšení databáze a dá se říct, že tahle knihovna představuje logiku celé aplikace.

Je závislá na dvou knihovnách, bez kterých by nemohla provést svoji činnost. Využívá knihovnu pro zpracování SQL dotazů pro získání listu s takovými dotazy, které už



Obrázek 11: Třídní diagram knihovny pro přístup k databázi

nejsou jen řetězce znaků, ale už se s nimi může pracovat jako s objekty, které poskytují jednotlivé části k dalšímu zpracovaní. Následně komunikuje s knihovnou pro přístup k databázi, což ji umožnuje komunikaci s databází.

5.3.3.1 Struktura

Můžeme zde nalézt mnoho částí, které poskytují funkce a jsou nezbytné pro provedení správného zmenšení databáze. Stejně jako v předchozích knihovnách budou mít části stejné jména jako třídy, které nalezneme v této knihovně:

- **DbSubsetterMS**

  Představuje nejhlavnější část v této knihovně. Řídí celý proces zmenšení od načtení vstupů až po provedení samotného zmenšení databáze.

- **DataSubsetManager**

  Řídí a kontroluje proces kopírování dat. Nejdříve inicializuje, jaké data a kam se mají zkopírovat a následně se postará, aby k tomu opravdu došlo za předpokladu, že je vše v pořádku. Kopírování dat se provádí pomocí transakce, takže pokud by nastala nějaká chyba, tak se vrátí databáze (zmenšená) do posledního konzistentního stavu.

- **DbAnalyzerMS**

  Slouží pro analyzování zdrojové databáze, ze které si vytáhne informace o všech tabulkách a schématech a následně tyto informace uloží do paměti a předá je části DbSubsetterMS.

- **TableObjectCreator**

  Vytváří objekty, které byly pojmenované jako „TableObject“. Takhle pojmenované byly proto, že každý takový objekt představuje „tabulku“, to znamená, že jde o objekt, který v sobě udržuje nejen informace o tabulce, ale ještě další věci, které jsou potřeba během procesu zmenšení databáze.

- **InsertDataQueryCreator**

  Vytváří objekty, které slouží jako „schránky“ pro uložení SQL příkazů typu INSERT. Tyhle příkazy pro vložení dat slouží k tomu, aby se následně provedli v databázi a došlo tedy ke zkopírování dat ze zdrojové databáze do cílové.

- **ScriptWriter**

  Vytváří výsledný skript, ve kterém můžeme nalézt všechny potřebné SQL příkazy, které jsou potřeba pro provedení zmenšení databáze. Tenhle skript se vždy nevytvoří, jenom pokud to nastavíme v grafickém rozhraní.

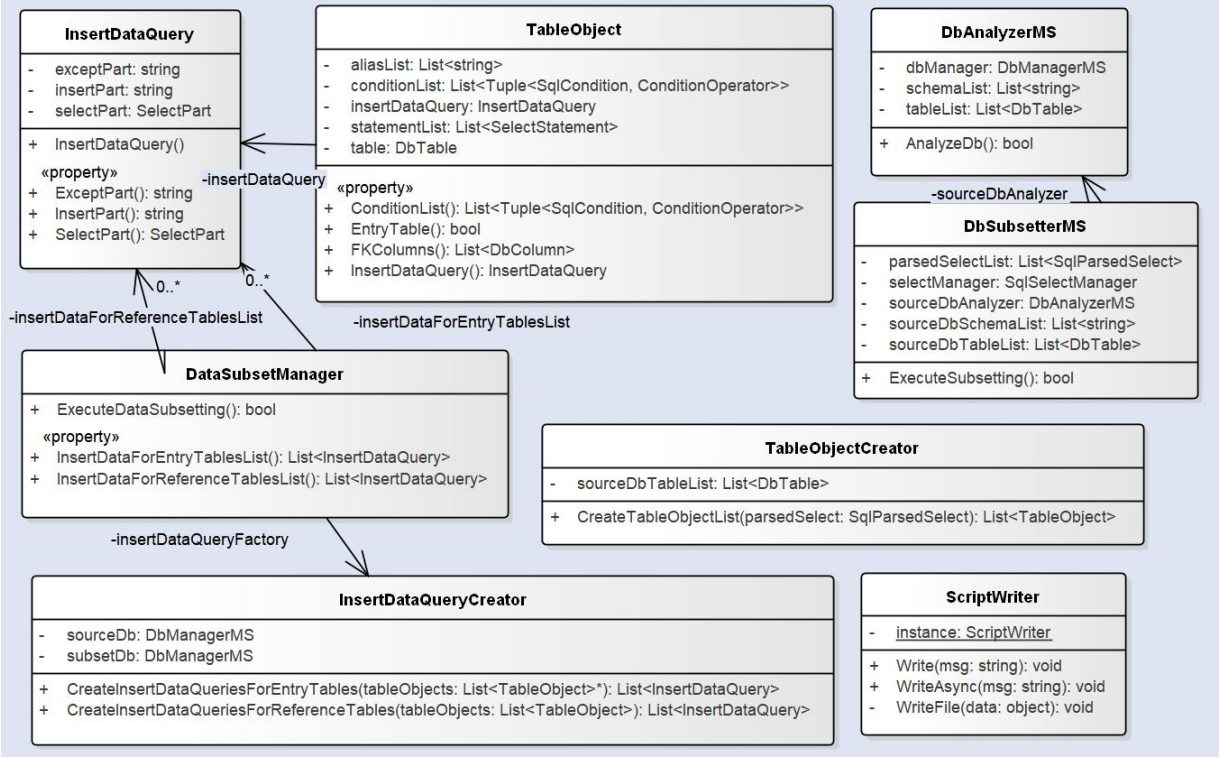
5.3.3.2 Vstupy a výstupy

Abychom mohli využít tuhle knihovnu, budeme muset k ní přistupovat přes třídu DbSubsetterMS, což jsme si už pověděli, že představuje hlavní část této knihovny a o vše se postará. Tudíž vstupy, které požaduje, jsou:

- řetězec připojení (angl. connection string) pro zdrojovou a zmenšenou databázi,
- cesta k souboru se vstupními SQL dotazy a
- znak, který odděluje jednotlivé vstupní SQL dotazy, v našem případě je to znak „;“.

Výstupy, které můžeme po provedení celého procesu očekávat, jsou:

- zmenšená data dle vstupních SQL dotazů v příslušných tabulkách, které se nacházejí ve zmenšené databázi a
- výsledný skript, který se vytvoří jenom, pokud to vyžadujeme.



Obrázek 12: Třídní diagram knihovny pro zmenšení databáze

5.3.4 GUI

Tahle část programu představuje grafické rozhraní, přes které uživatel komunikuje se samotným programem. Může nastavit různé věci jako: cestu ke vstupnímu souboru nebo zda se má vytvořit výsledný skript. Dále pak může provést samotné zmenšení databáze. Tuhle část pak zařazujeme do prezentační vrstvy.

Jedná se o desktop aplikaci, která je spustitelná pod operačním systémem Windows a její náhled můžeme vidět na obrázku 16

5.3.4.1 Části GUI

Vysvětlíme si jaké části GUI má a jaké jsou jejich funkce. Nejdříve se zaměříme na hlavní okno (viz obr. 16), kde najdeme hned několik částí:

- **Seznam tabulek ve zdrojové databázi**

  Tohle je spíše pro informaci, jaké tabulky se vyskytují ve zdrojové databázi. Tohle nám dává pocit toho, že v databázi jsou dané tabulky a můžeme tedy provádět zmenšení. Kdyby se neukázaly žádné tabulky, tak bychom asi těžko prováděli zmenšení a přivedlo nás by to k tomu, že máme například špatně zadanou databázi.

- **Vstupní SQL dotazy**

V téhle části si nastavíme cestu k souboru se vstupními dotazy. Dále tu máme možnost si otestovat vstupní dotazy, tím pak zjistíme, zda jsou správné a nemusíme počítat s tím, že zmenšení se neprovede úspěšně.

- **Konzole**

  Konzole slouží pro poskytování informací například o výsledku testu dotazu či provádění procesu zmenšení databáze.

- **Zmenšení databáze**

  Zde si můžeme nastavit, zda chceme vytvořit výsledný skript, pokud ano, tak se uloží do složky Skript v adresáři, kde se nachází exe soubor aplikace. Jako hlavní je tu možnost spustit samotný proces zmenšení.

Jako další podstatné okno, které je vhodné si popsat je okno k přihlášení se k databázi, jehož vzhled můžeme vidět na obrázku 15. Skládá se tedy z výběru dvou databází, kde:

- si vybereme, zda chceme vložit řetězec připojení (obsahuje vše, co je potřeba k připojení k databázi) nebo budeme postupně vyplňovat údaje o: databázovém serveru, jméno databáze, loginu a hesle,
- vybereme možnost, zda si tyto informace uložit, takže při příštím přihlašování už nemusíme znovu psát přihlašovací údaje,
- otestujeme si připojení k daným databázím a
- následně se připojíme a dostaneme se zpět do hlavního okna.

Důležitá poznámka pro přihlašovací okno je, pokud nezadáme správné přihlašovací údaje, tak se přes tlačítko Ok nedostaneme zpět do hlavního okna, ale budeme muset dát zpět a tudíž nebudeme přihlášení k databázím.

Poslední okno, které můžeme v aplikaci nalézt je okno O programu, kde už jen nalezneme nějaké informace (stručný popis aplikace, název aplikace).

5.3.4.2 Propojení s předchozími knihovnami

Jelikož touhle aplikací můžeme spustit zmenšení databáze, tak si musíme lehce vysvětlit, jak to provede. Ze začátku si nastavíme všechny vstupy a splníme všechny podmínky, které jsou potřeba pro uskutečnění spuštění procesu zmenšení databáze tyhle dvě věci:

- načtený vstupní soubor
- fungující připojení k oběma databázím

A když jsme tak učinili, tak tahle aplikace má přístup ke knihovně zmenšení databáze a z ní si vytvoří instanci objektu, která si převezme dané vstupy a provede celý proces. Tím pádem se tahle část aplikace (GUI) už nemusí o nic jiného starat a pak už jen čeká na výsledek, zda byl proces úspěšný či nikoliv.

5.4 Odchytávání chyb

Při spuštění a následném provádění zmenšení databáze mohou nastat komplikace a to takové, že se proces nedokončí správně. Může to být z mnoha důvodů, například když zadáme špatné vstupní SQL dotazy, které nejdou spustit ve zdrojové databázi, tak můžeme očekávat, že nebude fungovat ani na zmenšené databázi a zároveň celý proces zmenšení nemusí a ani nebude správně dokončen. Samozřejmě, že jsou možné i jiné důvody, proč se proces neprovede úspěšně, a k tomu pak aplikace poskytuje Loggery, což jsou v podstatě takoví zapisovatelé chyb, které byly zachycené a zapsané do logů (textový soubor), ve kterém můžeme najít stručné informace o chybách po případě i ve které části programu se staly. Z toho se můžeme dozvědět, co je za problém a mohli ho nějak řešit. Každá knihovna má takového „zapisovatele“.

6 Algoritmus zmenšení databáze

Tento algoritmus je možné použít pouze pro databáze, které se nacházejí na MS SQL Serveru. Logika algoritmu je rozložena ve třech knihovnách:

- knihovna pro zpracování SQL dotazů,
- knihovna pro přístup k databázi
- a knihovna pro zmenšení databáze.

Algoritmus probíhá v šesti krocích:

1. Testování vstupů
2. Zpracování vstupních SQL dotazů
3. Analýza zdrojové databáze
4. Inicializace cílové databáze
5. Vytvoření kolekce příkazů insert-select
6. Spuštění kolekce příkazů insert-select v cílové databázi

6.1 Testování vstupů

Vstupy se testují v knihovně pro zmenšení databáze. Nejdříve se testuje cesta k souboru se vstupními SQL dotazy. Pokud je cesta správná a soubor existuje, tak se pokračuje kontrolou připojení ke zdrojové i cílené databázi. Pokud vše bylo v pořádku, tak se pokračuje bodem 6.2. Kdyby nastala chyba v kontrole cesty nebo připojení, tak aplikace ukončí algoritmus zmenšení databáze a vrátí se varovná hláška: „Zmenšení databáze neproběhlo úspěšně!“ a v jednom ze tří errorlogů, které se nacházejí adresáři aplikace ve složce Logs, najdeme informaci o chybě, která nastala během tohoto testování vstupů.

6.2 Zpracování vstupních SQL dotazů

Veškerá logika zpracování se nachází v knihovně pro zpracování SQL dotazů. Ze začátku načteme vstupní SQL dotazy ze vstupu (souboru) tím, že celý obsah (řetězec znaků) v souboru přečteme a následně rozdělíme dle znaku „;“ na menší řetězce, které představují vstupní SQL Select. Pokud se ve vstupním souboru nacházeli řádky, které začínají řetězcem „–“ (řádkový komentář v SQL jazyce), tak se tyto řádky přeskočí. Rovněž se přeskočí znak „;“. Rozdělené menší řetězce se uloží do kolekce sqlQueryCollection. Tohle vše provedl objekt SqlLoader a použil metodu LoadFromFile.

Následně objekt SelectStringLexer metodou ParseTokens čte řetězce s sqlQueryCollection a rozděluje je na jednotlivé tokeny (slova). Tokeny získá tím, že se řetězec rozděluje pomocí mezer, čárek nebo levé či pravé kulaté závorce. Závorky i čárky se stanou tokenem, mezera nikoliv. Získané tokeny se uloží do kolekce tokenů (objektů SelectStringToken). Dostaneme tolik kolekcí tokenů kolik je vstupních SQL Selektů.

Objekt SelectStatementCreator, který použije metodu CreateSelectStatement, si načte jednotlivé kolekce tokenů a začne je zpracovávat. Zpracování tokenů probíhá tak, že se tokeny rozdělují do objektů: SqlSelect, SqlFrom, SqlWhere, SqlGroupBy, SqlHaving a SqlOrderBy. Všechny tyto objekty nalezneme v objektu SelectStatement.

Například chceme vytvořit objekt SqlSelect. Při přečtení tokenu (řetězce) „SELECT“ poznáme, že se bude jednat o klauzuli SELECT. Čteme další tokeny, pokud narazíme token „(“, tak si ji uložíme do dočasného listu závorek a když narazíme na token „)“, tak poslední přidanou levou závorku vymažeme v listu závorek. Tokeny čteme a ukládáme do dočasného listu, dokud tedy nenarazíme na token „FROM“ a zároveň list závorek bude prázdný. Po přečtení tokenu „FROM“ a list závorek je prázdný, čteme dočasný list tokenů, kde tokeny rozdělíme pomocí tokenů „,“ na menší listy tokenu a z nich už vytvoříme objekt SqlColumn (sloupec), který přidáme do objektu SqlSelect.

Pokud při vytváření sloupce narazíme na token „(“ a hned za ním na token SELECT, tak to znamená existenci poddotazu a ten je pak řešen rekurzivně tak, že byly zpracovány všechny tokeny mezi tokeny „(“ a „)“ a vytvořil se z nich objekt SelectStatement, který přestavuje sloupec ve formě poddotazu, který uložíme do listu sloupců v objektu SqlSelect. Stejným principem se vytvoří všechny ostatní objekty, které najdeme v objektu SelectStatement.

Pokud vstupní SQL Select má špatnou syntaxi, tak se zpracuje stejně jako SQL Select se správnou syntaxí. Výsledkem zpracování vstupních SQL dotazů je kolekce objektů SqlParsedSelect, který v sobě zahrnuje objekt SelectStatement.

6.3 Analýza zdrojové databáze

Použije se objekt DbAnalyzerMS a ten použije metodu AnalyzeDb k nalezení všech tabulek a schémat ve zdrojové databázi. Informace o tabulkách a schématech si uložíme do listů tableList a schémaList.

6.4 Inicializace cílové databáze

V tomhle kroku si inicializujeme cílovou databázi. Po analýze zdrojové databáze, máme k dispozici list všech tabulek a schémat ve zdrojové databázi. Po zpracování vstupních SQL dotazů máme kolekci objektů SqlParsedSelect. Objekt DbSubsetterMS použije metodu GetTableObjectListFromAllParsedSelects a vytvoří list objektů TableObjekt (tableObjectList). Z tohohle listu si pak lehce vytahneme seznam všech tabulek

a schémat, které následně pomocí objektu SubsetDbInicializer přes metodu InicializeSubsetDb zkopírujeme všechny schémata a tabulky s primárními i cizími klíči do cílové databáze. Tabulky v cílové databázi budou po vytvoření prázdné. Pokud by už tabulky existovaly v cílové databázi, tak se již nevytvoří a vytvoří se jen ty tabulky, které ještě neexistují.

6.5 Vytvoření kolekce příkazů insert-select

K vytvoření kolekce příkazů insert-select potřebujeme kolekci zpracovaných vstupních SQL dotazů (parsedSelectList), kterou jsme získali v bodě 6.2. Tyhle dotazy následně jednotlivě procházíme a vytváříme k nim množinu příkazů insert-select, které ukládáme do výsledné kolekce insertQueryList.

Množina příkazů insert-select k danému vstupnímu dotazu z kolekce parsedSelectList se vytváří tak, že:

1. se vytvoří kolekce tableObjectList
2. a následně se vytvoří množina příkazů insert-select

6.5.1 Vytvoření kolekce tableObjectList

Tuhle kolekci vytvoří objekt TableObjectCreator. Jako vstup je vyžadován vstupní SQL dotaz z kolekce parsedSelectList a výstup je kolekce tableObjectList, který obsahuje tolik prvků, kolik se nalezne tabulek ve vstupu plus všechny referenční tabulky, které mají na nalezené tabulky referenci. TableObjectList se vytváří:

1. Zjistí se tabulky, které se nacházejí ve vstupu
2. Najdou se všechny referenční tabulky
3. Hledají se aliasy tabulek a přiřadí se k tabulkám
4. Najdou se všechny podmínky a přiřadí k příslušné tabulce

6.5.1.1 Zjištění tabulek ve vstupu

Jelikož vstup je zpracovaný vstupní SQL dotaz, tak si z něho vytáhneme seznam tabulek a vytvoříme ke každé tabulce v seznamu tableObject a uložíme si ho do tableObjectList.

6.5.1.2 Nalezení referenčních tabulek

Když jsme analyzovali zdrojovou databázi 6.3, tak jsme získali seznam tabulek tableList, kde každá tabulka obsahuje informace o referenčních tabulkách. Pak pro každý

objekt v tableObjectList nacházíme referenční tabulky tak, že v tableList najdeme příslušnou tabulku, která se shoduje s tabulkou v tableObject. Pokud najdeme referenční tabulku, která se ještě nenachází v tableObjectList, tak pro ni vytvoříme tableObject.

Tímhle způsobem jsme našli všechny tabulky, které budou potřeba ke správnému kopírování dat.

6.5.1.3 Hledání aliasů ve vstupu

Projde se vstup a hledají se všechny aliasy a přiřadí se k příslušným tabulkám v tableObjectList.

6.5.1.4 Hledání všech podmínek ve vstupu

1. Procházíme všechny SELECT příkazy, které vstup obsahuje.
2. Když narazíme na klauzuli WHERE, ON či HAVING, tak v nich procházíme podmínky.
3. Podmínku analyzujeme a získáme z nich informace: nazve sloupců (s prefixy), predikát, zda obsahuje vnořený dotaz nebo se před podmínkou vyskytuje logická spojka (AND, OR).
4. Analyzovanou podmínku spolu s informacemi přiřadíme k příslušným tabulkám v tableObjectList dle prefixu či názvu sloupce nalezeného v podmínce.

Tímto jsme získali tableObjectList, jehož prvek je objekt typu TableObject, který obsahuje:

- informace o tabulce (název, sloupce, prim. a cizí klíče, ref. tabulky)
- aliasy tabulky
- existence tabulky ve vstupu
- SQL Select ze vstupu, kde se tabulka nachází
- list podmínek

6.5.2 Vytvoření množiny příkazů insert-select

Množinu těchto příkazů vytvoří objekt DataSubsetManager. Ten bere vstup kolekci tableObjectList a výstup dává kolekci insertQueryList, který vznikne tak, že se postupně prochází vstup a pro každý tableObject se vytvoří příkaz insert-select.

Příkaz insert-select se vytvoří ze tří částí:

1. InsertPart
2. SelectPart
3. ExceptPart

6.5.2.1 InsertPart

Tahle část příkazu tvoří řetězec ve formě: INSERT INTO název tabulky v cílené databázi. Jedná se o část, která nám říká, že příkaz bude vkládat data.

6.5.2.2 SelectPart

Jedná se o SQL Select, který vybírá množinu dat ze zdrojové databáze. SelectPart vzniká tak, že jako vstup máme tableObject a výstup je SQL Select řetězec, který se vytvoří:

1. Vytvoříme dvě základní klauzule dotazu: SELECT a FROM a získáme řetězec: SELECT * FROM tabulka alias. Název tabulky a aliasu najdeme ve vstupu.
2. Vytvoříme klauzuli WHERE:
   2.1. Procházíme list podmínek ve vstupu.
   2.2. Pokud je podmínka obsahuje pouze sloupec, predikát a jednoduchý výraz například: s.jmeno LIKE 'P%', tak z podmínky vytvoří řetězec a uloží se do klauzule WHERE.
   2.3. Pokud je podmínka složitější (vnořený dotaz), tak se zpracuje následovně. Když se jedná o nezávislý poddotaz, tak se prochází a analyzuje. Pokud se vyskytuje před poddotazem predikát (IN, ANY a další), tak se v poddotazu změní v klauzuli SELECT název sloupce tak, aby odpovídal se sloupcem, který je v podmínce před predikátem. Jestli se jedná o závislý poddotaz, tak se nalezne podmínka v poddotazu, která dělá poddotaz závislým a to tak, že podmínka bude obsahovat alias či název sloupce, který neodpovídá žádné tabulce v tom závislém poddotaze. Z něho se zjistí, na jaké tabulce je poddotaz závislý. K takové tabulce se pak přidá tahle podmínka se závislým poddotazem v podobném principu jako u podmínky s nezávislým poddotazem (úprava SELECT klauzule poddotazu za příslušný sloupec klíče, aby se podmínka dala přidat k tabulce, ke které je závislý). Pak se s podmínek vytvoří řetězce a vloží se do klauzule WHERE.
3. Pokud existuje klauzule HAVING dle přítomnosti podmínky, která byla nalezena v HAVING klauzuli, tak se vytvoří podmínka, která obsahuje predikát EXISTS a závislý vnořený dotaz, který se vytvoří tak, že známe podmínku pro HAVING klauzuli a známe sloupce, dle kterých se Select seskupoval, tyhle sloupce vložíme do

klauzule SELECT a GROUP BY. „Having“ podmínku vložíme do klauzule HAVING a na konci ještě přidáme podmínku, která propojí závislý poddotaz s tabulkou dle klíčů. A tuhle vytvořenou podmínku vložíme jako řetězec do klauzule WHERE.

Pokud se podmínek více, tak se mezi podmínky vkládají logické spojky (AND, OR), ty jsou obsaženy v podmínkách, takže pokud daná podmínka obsahuje logickou spojku, tak se v řetězci klauzule WHERE objeví také.

6.5.2.3 ExceptPart

Poslední část příkazu insert-select. Jedná se o řetezec: EXCEPT SELECT * FROM tabulka v cílené databázi. Tabulka je shodná s tabulkou v InsertPart. Tahle část zajišťuje, že nedojde k vložení již existujících záznamů v cílové tabulce.

Takhle se zpracuje každý objekt v tableObjectList a všechny vytvořené příkazy insert-select se uloží do kolekce insertQueryList.

6.6 Spuštění kolekce příkazů insert-select v cílové databázi

Jako poslední krok se všechny vytvořené příkazy insert-select spustí na cílové databázi a tím dojde ke zkopírování dat se zdrojové do cílové databáze a tedy vytvořená cílová databáze.

6.7 Ukázka algoritmu na příkladech

Příklad 1

Mějme na vstupu 1 vstupní SQL dotaz, který vybírá takové výrobce a jejich adresy, kteří se nacházejí ve státu začínající na velké počáteční písmeno A až E a zároveň mají zboží, které nemají skladem. Vstupní SQL dotaz obsahuje nezávislý poddotaz a spojení. Zdrojová databáze je 14.

```
select *
from ES_Zbozi.Vyrobce v
inner join ES_Zamestnanec.Adresa a on v.IDadresa = a.IDadresa
where a.stat LIKE '[A-E]%' AND
v.IDvyrobce = Any( select distinct z.IDvyrobce
                   from ES_Zbozi.Zbozi z
                   inner join ES_Zbozi.Sklad s on z.IDzbozi = s.IDzbozi
                   where s.pocetKusuNaSklade = 0);
```

Výpis 9: Vstupní SQL dotaz pro příklad 1

Algorimtus v kroku 6.1 zkontroluje vstupy a pokud jsou v pořádku, tak se pokračuje dále, jinak se algoritmus zastaví. V dalším kroku 6.2 se vstupní dotaz zpracuje a vytvoří se objekt SqlParsedSelect. Hned potom se v kroku **??** a 6.4 analyzuje zdrojová databáze a inicializuje se cílová databáze. Byly tedy nalezeny všechny potřebné schémata, tabulky i klíče, které byly následně zkopírovány do cílové databáze. Všechny nově vytvořené tabulky jsou prázdné. Vytvořili se celkem 4 tabulky: Adresa, Vyrobce, Zbozi a Sklad.

V kroku 6.5 vytvoří kolekce příkazů insert-select, kde si například můžeme ukázat příkaz vložení dat do tabulky Výrobce. Můžeme si všimnout, že v částech INSERT a EXCEPT se nachází tabulky v cílové databázi a v SELECT části jsou tabulky ze zdrojové databáze.

```
INSERT INTO Eshop_subset.ES_Zbozi.Vyrobce
SELECT * FROM Eshop.ES_Zbozi.Vyrobce v
WHERE v.IDvyrobce =Any (SELECT distinct z.IDvyrobce
                FROM ES_Zbozi.Zbozi z
                inner join ES_Zbozi.Sklad s
                ON z.IDzbozi  = s.IDzbozi
                WHERE s.pocetKusuNaSklade = 0 )
AND v.IDadresa IN ( SELECT a.IDadresa
              FROM Eshop.ES_Zamestnanec.Adresa a
              WHERE a.stat LIKE '[A-E]%' )
EXCEPT
SELECT * FROM Eshop_subset.ES_Zbozi.Vyrobce
```

Výpis 10: Příkaz insert-select pro tabulku Vyrobce

V posledním kroku 6.6 se zkopírují data do všech vytvořených tabulek v cílové databázi tak, že se spustí všechny příkazy v kolekci příkazů insert-select.

Příklad 2

Zdrojová databáze je stejná jako v příkladu 1. Teď budeme mít vstupní SQL dotaz, který obsahuje klauzuli HAVING. Tenhle dotaz vrací zákazníky, kteří mají méně než 4 objednávky vytvořené po 1. lednu 2017.

```
SELECT o.IDzakaznik, count(o.IDobjednavka)
FROM [ES_Objednavka].[Objednavka] o
where o.datumVytvoreni <= '1.1.2017'
GROUP BY o.IDzakaznik
HAVING count(o.IDobjednavka) < 4;
```

Výpis 11: Vstupní SQL dotaz pro příklad 1

Algoritmus zmenšení databáze pracuje stejně jako v příkladu 1. Zajímavé pro nás je to, jak se zpracovala podmínka v klauzuli HAVING. Ukážeme si to na jednom z vytvořených insert-selectů.

```
INSERT INTO Eshop_subset.ES_Zakaznik.Zakaznik
SELECT * FROM Eshop.ES_Zakaznik.Zakaznik
WHERE IDzakaznik IN ( SELECT IDzakaznik
                FROM Eshop.ES_Objednavka.Objednavka o
                WHERE o.datumVytvoreni <= '1.1.2017'
                AND EXISTS (SELECT o1.IDzakaznik , count(o1.IDobjednavka )
                        FROM Eshop.ES_Objednavka.Objednavka o1
                        WHERE o1.datumVytvoreni <= '1.1.2017'
                        AND o.IDzakaznik = o1.IDzakaznik
                        GROUP BY o1.IDzakaznik
                        HAVING count(o1.IDobjednavka ) < 4))
EXCEPT
SELECT * FROM Eshop_subset.ES_Zakaznik.Zakaznik
```

Výpis 12: Příkaz insert-select pro tabulku Zakaznik

Vidíme, že se podmínka v HAVING klauzuli se přesunula do závislého vnořeného dotazu a podmínka o.IDzakaznik = o1.IDzakaznik nám zajišťuje, že poddotaz filtruje data patřící tabulce Objednávka o.

Příklad 3

Opět pracujeme se zdrojovou databází jako v příkladu 1. V tomhle příklady si ukážeme, jak se řeší cizí klíče. Na vstupu budeme mít triviální vstupní SQL dotaz, který nám vybere záznamy z tabulky Ucet, které byly vytvořeny po začátku roku 2018:

```
SELECT *
FROM ES_Zakaznik.Ucet
WHERE datumVytvoreni < '01.01.2018'
```

Výpis 13: Triviální vstupní SQL dotaz pro tabulku Ucet

Algoritmus zmenšení databáze proběhne stejně jako v příkladu 1. Jako výsledek dostaneme tyto příkazy insert-select, které spustíme v cílené databázi v následujícím pořadí.

```
INSERT INTO Eshop_subset.ES_Zakaznik.Zakaznik
SELECT * FROM Eshop.ES_Zakaznik.Zakaznik z
WHERE z.IDzakaznik IN ( SELECT u.IDzakaznik
                FROM Eshop.ES_Zakaznik.Ucet u
```

```
                WHERE u.datumVytvoreni < '01.01.2018' )
EXCEPT
SELECT * FROM Eshop_subset.ES_Zakaznik.Zakaznik
```

Výpis 14: Příkaz insert-select pro tabulku Zakaznik

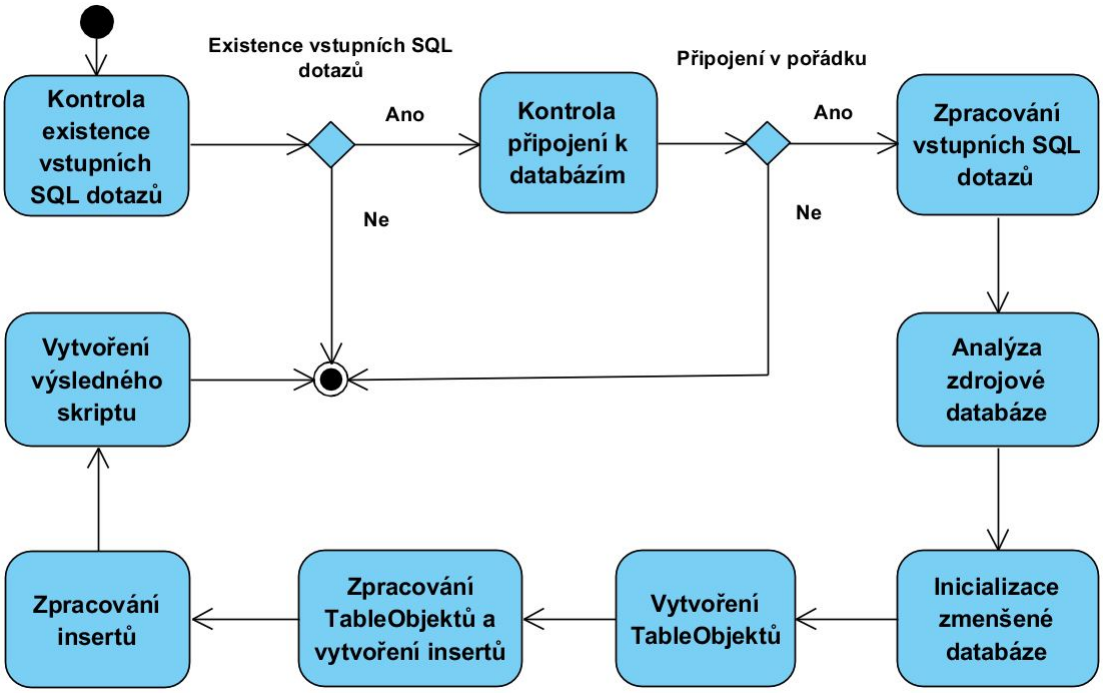
```
INSERT INTO Eshop_subset.ES_Zakaznik.Ucet
SELECT *
FROM Eshop.ES_Zakaznik.Ucet
WHERE datumVytvoreni < '01.01.2018'
EXCEPT
SELECT * FROM Eshop_subset.ES_Zakaznik.Ucet
```

Výpis 15: Příkaz insert-select pro tabulku Ucet

Při pohledu na schéma zdrojové databáze 14 vidíme, že tabulka Ucet má cizí klíč odkazující na tabulku Zakaznik. V příkazu, který je zobrazený ve výpisu 14, vidíme, že se tento cizí klíč řeší dodatečným příkazem insert-select, kde vytvoříme podmínku, do které vložíme primární klíč z tabulky Zakaznik a přidáme predikát IN s vnořeným dotazem, ve kterém bude SELECT příkaz, který vybírá množinu záznamů z tabulky Ucet.

6.8 Grafické znázornění algoritmu

Algoritmus si taky můžeme zobrazit pomocí diagramu 13, na kterém můžeme si vizuálně zobrazit jednotlivé kroky a jejich pořadí, ve kterých se provádějí.



Obrázek 13: Diagram aktivit logiky zmenšení databáze

7 Testování

V téhle kapitole se budeme věnovat testování naší implementované aplikaci. Aplikace bude otestována tak, že zmenšíme zdrojovou databázi a získáme cílovou databázi, která bude představovat výsledek.

Testování bude prováděno na databázích na lokálním serveru a běžném osobním počítači.

7.1 Vstupy

K tomu, abychom mohli provést testování, budeme potřebovat vstupy:

- zdrojová a cílová databáze (MS SQL Server)
- vstupní SQL dotazy, které jsou spustitelné ve zdrojové databázi

7.1.1 Popis

Při tvorbě této databáze jsme se inspirovali již existující databázi, která se jmenuje „AdventureWorks 2014“. Ta například mohla sloužit jako vstup, ale pro náš účel jsme pak zvážili, že si vytvoříme vlastní, která bude v pár ohledech podobná a bude nám obstojně sloužit k našemu testování.

V naší zdrojové databázi se vyskytuje celkem 17 tabulek, které jsou navíc v různých schématech, a celkově obsahuje cca 17 milionů záznamů.

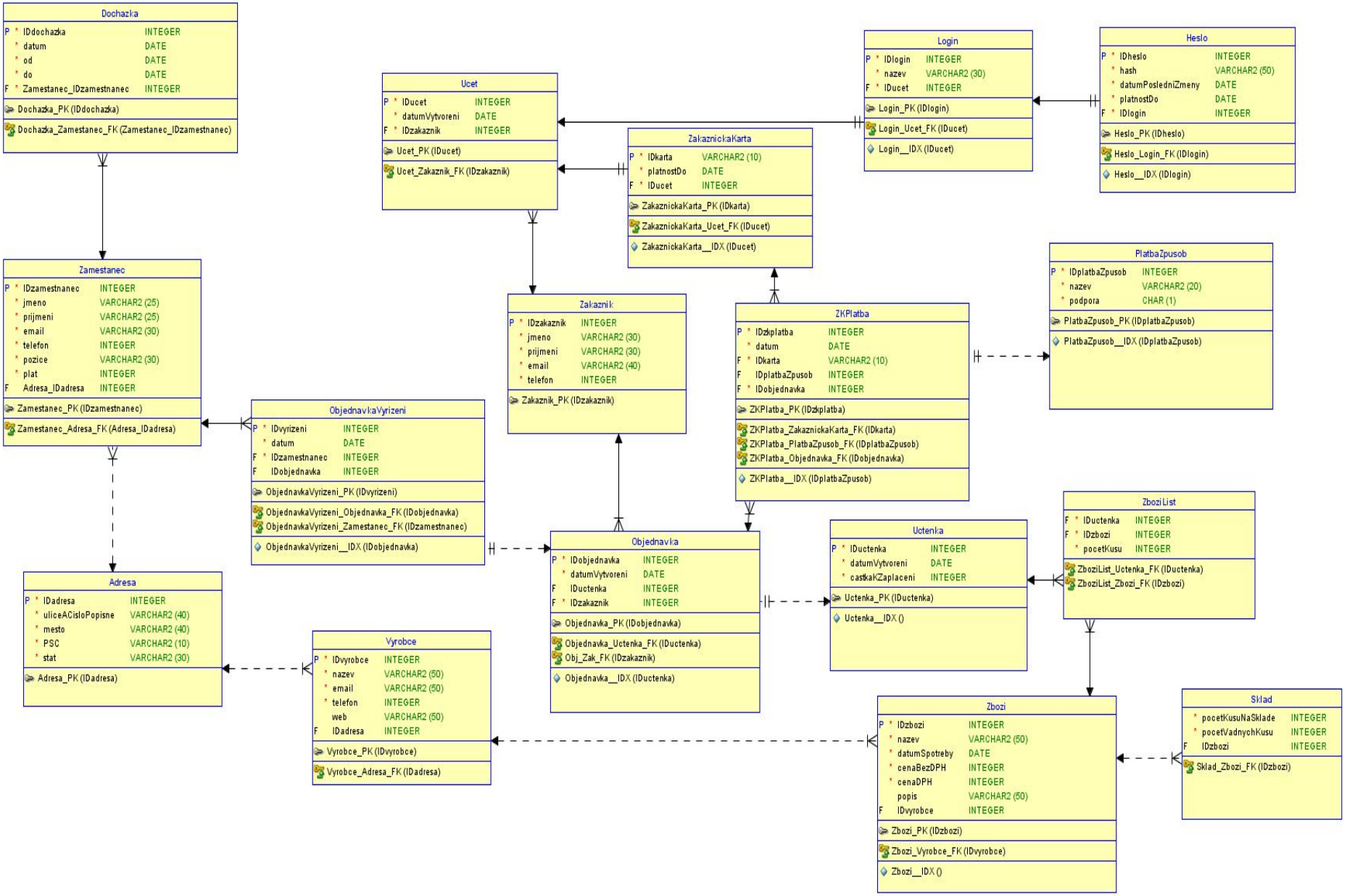
Co se týče ostatních objektů, tak v této databázi nám postačují pouze tabulky, žádné jiné objekty jako například: indexy, pohledy nebo spouštěče jsme neřešili. Mohou se tam objevit, ale jsou irelevantní pro naše testování, protože netestujeme funkcionalitu nebo rychlost dotazování, ale jen kopírujeme tabulky a data.

Na obrázku 14 můžeme detailněji vidět, jak jsou jednotlivé tabulky propojené. Dané relační schéma bylo vytvořeno v nástroji Oracle Datamodeler a zároveň sloužilo jako zdroj pro skript, pomocí kterého jsme vytvořili tabulky.

7.2 Vstupní SQL dotazy

Tyto dotazy jsou jedny z nejdůležitějších částí při testování. Právě ony zaručují dané zmenšení. Na nich to záleží nejvíce, protože pokud bychom měli dotazy, které vyberou všechny záznamy z daných tabulek, tak se zkopírují všechny záznamy z těch tabulek, ale pokud máme takové dotazy, které vyselektují jen určitou část dat, tak se následně zkopírují jen ta určitá část a můžeme mluvit o nějakém zmenšení.

V našem případě jsme použili jen několik dotazů, které mimochodem můžete najít v příloze, kde se na ně můžete detailněji prozkoumat. Volili jsme takové dotazy, které



Obrázek 14: Relační schéma zdrojové databáze

v sobě zahrnují některé „složitější“ struktury, jako například vnořené dotazy nebo přítomnost spojení či klauzule HAVING.

Přestavte si spíše to vybírání dotazů z pohledu toho, že se nejedná pouze o triviální dotazy, ale že aplikace dokáže zpracovat i trochu složitější čili netriviální dotazy a provést pomocí nich zmenšení databáze.

7.3 Použitý stroj na testování

Testování proběhl na osobním notebooku, jehož hardwarové parametry můžete vidět v tabulce 4.

Tabulka 4: Parametry použitého notebooku

Model	Lenovo Y50-70 Touch
Operační systém	Windows 7 Professional
Procesor	Intel Core i7-4710HQ 2,5Ghz
Operační paměť	16GB DDR3 1600 MHz
Pevný disk	1TB 5400 ot/min

7.4 Výsledky testování

Provedli jsme testování tím, že jsme spustili aplikaci, následně jsme nahráli vstupní SQL dotazy a spustili test. Test se provedl úspěšně a dostali jsme následující výsledky, které si ukážeme v tabulkách a následně i popíšeme.

Tabulka 5: Přehled všech tabulek a výsledků zmenšení

Název tabulky	**Název schématu**	**Počet záznamů ve zdrojové databázi**	**Počet záznamů ve zmenšené databázi**	**Zmenšení tabulky [%]**	**Použití tabulky při zmenšení**
Objednavka	ES_Objednavka	1 898 481	410 845	78,4	ANO
ObjednavkaVyrizeni	ES_Objednavka	1 727 617	0	100	NE
Uctenka	ES_Objednavka	1 898 481	418 378	78	ANO
Heslo	ES_Zakaznik	200 000	0	100	NE
Login	ES_Zakaznik	200 000	0	100	NE
PlatbaZpusob	ES_Zakaznik	2	0	100	NE
Ucet	ES_Zakaznik	200 000	70 610	64,7	ANO
ZakaznickaKarta	ES_Zakaznik	200 000	70 610	64,7	ANO
Zakaznik	ES_Zakaznik	200 000	152 176	23,9	ANO
ZKPlatba	ES_Zakaznik	1 898 481	0	100	NE
Adresa	ES_Zamestnanec	100 996	39 138	61,25	ANO
Dochazka	ES_Zamestnanec	949 157	10 291	98,92	ANO
Zamestnanec	ES_Zamestnanec	100 000	9 717	90,3	ANO
Sklad	ES_Zbozi	3 243	815	74,87	ANO
Vyrobce	ES_Zbozi	996	915	8,13	ANO
Zbozi	ES_Zbozi	3 244	3 057	5,76	ANO
ZboziList	ES_Zbozi	7 602 202	8 428	99,88	ANO

V tabulce 5 jsme si uvedli pro přehled všechny tabulky. V prvních třech sloupcích vidíme název tabulky, schémata, ve kterých se tabulky nacházejí, a počty záznamů. Tyto informace nám říkají, co se vyskytuje ve zdrojové databázi.

Zbylé sloupce nám zobrazují výsledky, které jsme dostali během zmenšení databáze. V prvé řadě vidíme, kolik záznamů se z každé tabulky přesunulo do nové (zmenšené databáze), pak vidíme, o kolik procent se daná tabulka zmenšila a úplně v pravém sloupci vidíme, jaké tabulky byly použity, neboli zkopírované do zmenšené databáze.

Tabulka 6: Vybrané celkové hodnoty obou vstupních databází

Databáze	**Celkový počet záznamů**	**Počet tabulek**	**Velikost [MB]**
zdrojová	17 182 900	17	580,88
zmenšená	1 194 980	12	70,31

Z hodnot, které máme v tabulce 6, jsme zjistili, že databáze, pokud budeme mluvit o datech, se zmenšila cca o 93%. Jinak řečeno zmenšená databáze obsahuje 6,95% dat ze zdrojové databáze.

Pokud by nás zajímala časová stránka testování, tak se můžeme podívat v tabulce 7, kde najdeme časy pro jednotlivé části procesu zmenšení.

Tabulka 7: Naměřené časy během při zmenšení databáze

Zpracování vstupních dotazů	**Analýza zdrojové databáze**	**Inicializace zmenšené databáze**	**Kopírování dat**	**Celková čas zmenšení**
174ms	853ms	424ms	25s 505ms	27s 25ms

7.5 Nepotřebná záznamy

Po té, co jsme viděli výsledky testování, nás může napadnout, zda všechny záznamy, které byly zkopírovány, jsou potřebné a nestalo se tedy, že by se zkopírovali i záznamy, které nepotřebujeme, tudíž bychom měli ve zmenšené databázi i nějaké ty záznamy navíc.

Těžce se to dokazuje, když vstupních SQL dotazů je více než jeden, ale můžeme si to ukázat pro jeden dotaz. Vybereme si dotaz č. 3, který najdeme v příloze. Pak provedeme zmenšení a dostaneme zmenšenou databázi, ve které budou tři tabulky (viz tabulka 8).

Tabulka 8: Tabulky a data po zmenšení pomocí jednoho SQL dotazu

Název tabulky	**Počet záznamů**
Objednavka	190 658
Uctenka	190 658
Zakazník	15 225

Když se podíváme na vstupní SQL dotaz, tak když ho provedeme ve zdrojové i zmenšené databázi, tak dostaneme 15 225 zákazníků a k nim příslušné počty jejich objednávek. Takže tabulky Zakaznik a Objednavka potřebujeme, to že v objednávce se pak nachází 190 658 záznamů je proto, že tyhle záznamy jsou spojené referenci na záznamy zákazníků a ano, pokud filtrujeme objednávky dle podmínky ve WHERE i HAVING klauzuli, tak dostaneme právě tento počet objednávek. Ale co ta tabulka Uctenka? Ta se taky vytvoří, protože tabulky Objednavka a Uctenka mají mezi sebou vazbu (referenci), tak kvůli tomu, abychom mohli vložit záznamy do tabulky Objednavka, tak se musí vložit i odpovídající záznamy do tabulky Uctenka, vše je tedy způsobeno referencemi mezi tabulkami.

8 Závěr

Testování proběhlo v pořádku. Provedli jsme kontrolu správnosti výsledků tak, že jsme spustili všechny vstupní SQL dotazy na zdrojové i cílové databázi a dostali jsme stejné výsledky. Dále jsme provedli kontrolu, zda se ve zmenšené databázi nevyskytují záznamy navíc. Tahle kontrola byla zdlouhavá a museli jsme provádět jednotlivé dotazy zvlášť, jenom tehdy jsme si mohli být jistí, že se opravdu zkopírovali jen žádoucí záznamy. Tedy nebyly nalezeny žádné záznamy navíc. Tím si dovoluji říct, že testování proběhlo úspěšně.

Během jiných testování jsme se setkali s několika problémy, kdy nástroj nefunguje správně. Vytvořený nástroj neumí řešit všechny vstupní SQL dotazy, nalezneme tedy plno omezení. Vstupní SQL dotazy nesmí být příliš složité nebo obsahovat nějaké složitější konstrukce. Nástroj umí tedy zpracovat sice netriviální dotazy, ale ne zdaleka všechny. Dotazy, které nejsou spustitelné na zdrojové databázi, tak už jen to je předpoklad, že nástroj nebude fungovat správně. Navíc obě vstupní databáze musí být na jedné instanci a musí být na MS SQL Serveru, jinak opět nebude nástroj správně fungovat. Proto se domníváme, že existuje mnoho příčin, kdy nástroj neprovede zmenšení úspěšně.

Tímto nástrojem se nám podařilo vyřešit několik komplikovanějších úloh. Dokážeme zpracovat dotazy obsahující komplikovanější struktury (vnořené dotazy), spojení či klauzuli HAVING.

Povedlo se tedy vytvořit nástroj, který je schopný provést zmenšení databáze (subsetting) s ohledem na vstupní typické vytížení. Jedná se o originální řešení, přičemž žádný z existujících nástrojů není schopen takovéto funkcionality.

Literatura

[1] Chamberlin, Donald (2012). "Early History of SQL". IEEE Annals of the History of Computing. 34 (4): 78–82. Retrieved 3 February 2018 [online]. Dostupné na: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6359709

[2] Chamberlin, Donald D; Boyce, Raymond F (1974). "SEQUEL: A Structured English Query Language". Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control. Association for Computing Machinery: 249–264 [online]. Dostupné na: http://www.almaden.ibm.com/cs/people/chamberlin/sequel-1974.pdf

[3] University of Helsinki, Department of Computer Science, History of SQL [online]. Dostupné na: https://www.cs.helsinki.fi/u/laine/tuelip/sql_material/sql_history.html

[4] SQL Course [online]. [cit. 2018-03-05]. Dostupné na: http://www.sqlcourse.com/intro.html

[5] Databázové a znalostní systémy, Standardní příkazy jazyka SQL [online]. Dostupné na: http://www1.fs.cvut.cz/cz/u12110/dzs/

[6] GRUBER, Martin. Mistrovství v SQL. Praha: Softpress, c2004. ISBN 80-864-9762-3.

[7] C.L. Moffatt, Visual Representation of SQL Joins [online]. Dostupné na: https://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

[8] HENDERSON, Ken. Mistrovství v Transact-SQL. Praha: Computer Press, 2000. Profi. ISBN 80-722-6393-5.

[9] Toad for Oracle, User guide [online]. Dostupné na: http://in.bgu.ac.il/sharon/Documents/Toad.pdf

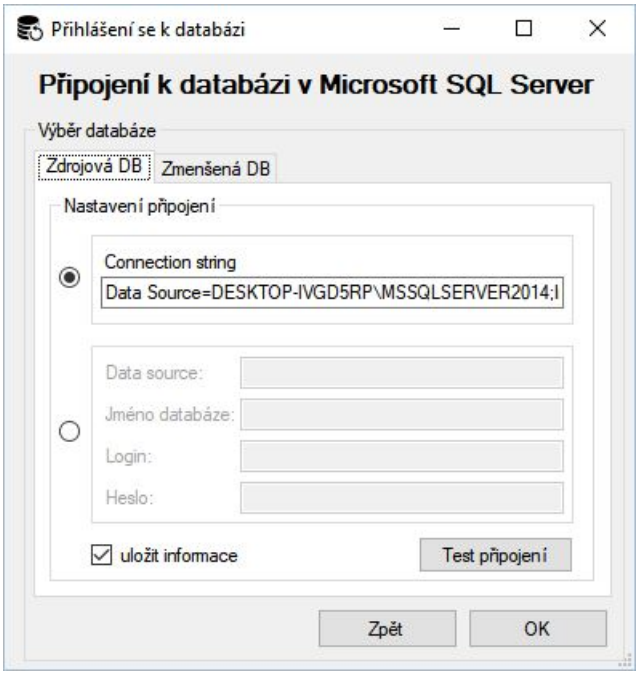
[10] Jailer, Database Subsetting Tool [online]. Dostupné na: http://jailer.sourceforge.net/

[11] The DataBee Software Manual, Applications Overview [online]. Dostupné na: https://www.databee.com/DTB/Help/dtb_Applications.htm

A Návod ke GUI

Ukážeme si jeden jednoduchý scénář, jenž může sloužit jako návod k obsluze GUI.

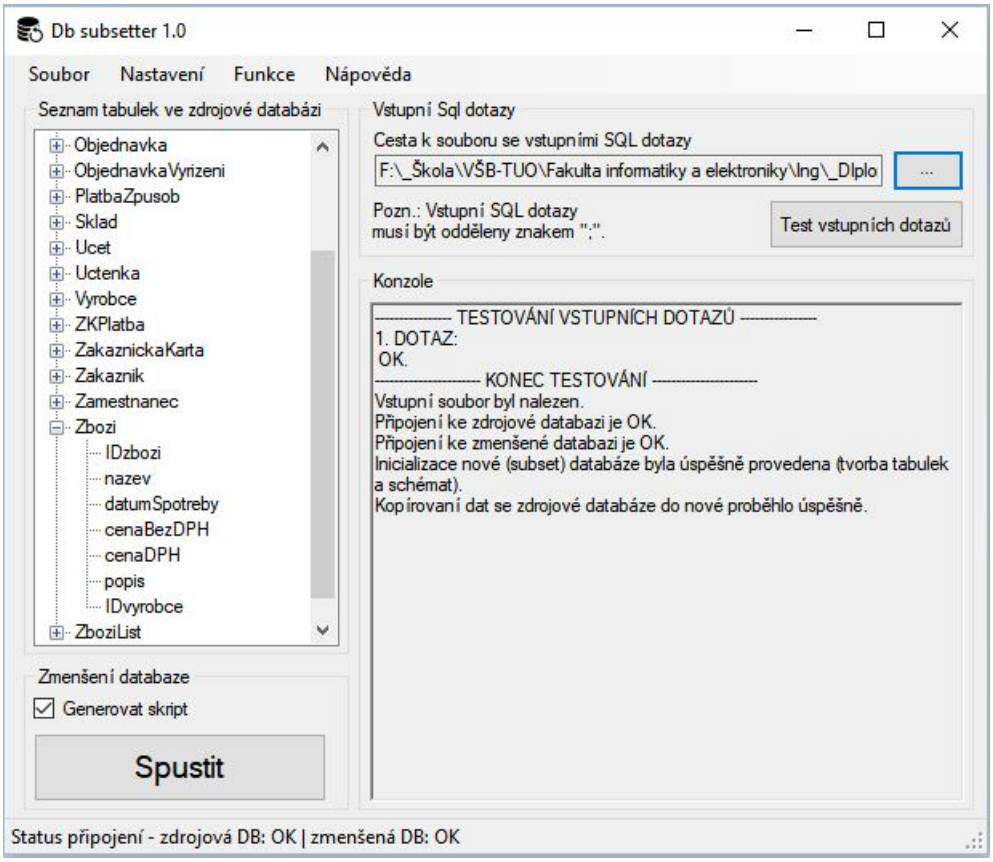
1. Po spuštění aplikace na nám otevře hlavní okno s oknem pro přihlášení k databázím, vyplníme požadované informace a klikneme na ok.



Obrázek 15: Přihlašovací okno

2. V hlavním okně si pak nastavíme cestu k souboru, kde se nacházejí vstupní SQL dotazy. Hned potom si můžeme otestovat nahrané vstupní SQL dotazy a tím zjistíme, zda jsou v pořádku a můžeme je použít. Dále si nastavit, zda chceme si uložit výsledný skript.

3. Až budeme mít všechno nastavené, tak můžeme spustit proces zmenšení databáze. Mějme na vědomí, že pokud nesplníme všechny podmínky, které jsou zmíněné v kapitole 5.3.4.2, tak nemůžeme proces spustit.



Obrázek 16: Hlavní okno aplikace

B Přehled vstupních SQL dotazů

Tyhle dotazy byly použity během testování naší aplikace.

1. SELECT COUNT(*) FROM ES_Zakaznik.Zakaznik
   WHERE IDzakaznik IN (SELECT IDzakaznik
   FROM ES_Zakaznik.Ucet
   WHERE datumVytvoreni like '%199%' AND IDucet = ANY (SELECT iducet
   FROM ES_Zakaznik.ZakaznickaKarta
   WHERE platnostDo <= '11.10.2008'));

2. SELECT o.IDzakaznik, count(o.IDobjednavka)
   FROM [ES_Objednavka].[Objednavka] o
   where o.datumVytvoreni <= '1.1.2017'
   GROUP BY o.IDzakaznik
   HAVING count(o.IDobjednavka) < 4;

3. select z.IDzakaznik, z.jmeno, z.prijmeni, count(*)
   from [ES_Zakaznik].[Zakaznik] z join [ES_Objednavka].[Objednavka] o on z.IDzakaznik = o.IDzakaznik
   where z.email like '%.com' and SUBSTRING(z.telefon,1,3) between '400' and '800'
   group by z.IDzakaznik, z.jmeno, z.prijmeni
   having count(*) > 8
   order by z.IDzakaznik;

4. select za.IDzamestnanec, za.jmeno, za.prijmeni
   from ES_Zamestnanec.Zamestnanec za
   inner join ES_Zamestnanec.Dochazka do on za.IDzamestnanec = do.IDzamestnanec
   inner join ES_Zamestnanec.Adresa ad on za.IDadresa = ad.IDadresa
   where ad.mesto like '[K-P]%' and do.datum > '31.06.2017';

5. select * from ES_Zbozi.Vyrobce v inner join ES_Zamestnanec.Adresa a on v.IDadresa = a.IDadresa
   where a.mesto = 'Czech Republic' OR a.mesto = 'Slovakia'
   OR v.IDvyrobce = Any(select distinct z.IDvyrobce
   from ES_Zbozi.Zbozi z inner join ES_Zbozi.Sklad s on z.IDzbozi = s.IDzbozi
   where s.pocetVadnychKusu = 0);

6. SELECT * FROM ES_Zbozi.ZboziList zb
   WHERE EXISTS (SELECT *
   FROM ES_Objednavka.Uctenka u
   WHERE zb.IDuctenka = u.IDuctenka AND u.castkaKZaplaceni < 150);

C Příloha na CD/DVD

Součástí BP/DP je CD/DVD.
Adresářová struktura přiloženého CD/DVD:

- Diplomová práce
  - Aplikace
    - Eshop_testovaci_DB - složka, která osbahuje zdrojovou databázi, data a skripty
    - DatabaseSubsetting - složka, která obsahuje implementovanou aplikaci v C#
  - Text - složka obsahuje diplomovou práci ve formátu pdf
    - Diplomová_práce.pdf
  - Readme.txt - soubor obsahující dodatečné informace